

07/12/00
16:59:14

```
// =====
// File Name      : $RCSfile: ba_cmu_core.v,v $
// Version        : $Revision: 1.84 $
// Status         : In beta, needs a few features, more testing
// Module         : Control Memory Unit (ba_cmu)
//
// Author         : Andrew Ryan
// E-mail         : acryan@wpinfo.soft.net, acryan@celox.com
// Phone         : USA, 1-314-615-6338
// Company        : Celox Communication Corporation
// Creation Date   : August 30, 1999
// Last Modified by : $Author: celox $
// Last Modified on : $Date: 2000/07/12 21:59:14 $
//
// Dependencies   : ba_cmu_port_dp
//
// Description    : The Control Memory Unit (CMU) transfers control cells
//                  to and scheduled cells from the PPU's. It must make
//                  sure that the FIFOs on one side do not underflow/
//                  overflow, and that the bandwidth to the dual-ported
//                  SRAMs is used optimally so that one misbehaving PPU
//                  does not adversely affect the other PPU's connected to
//                  this unit.
//
// Simulator      : VCS and Verilog-XL
//
// Simulation Notes : Here's the a diagram of the test bench:
//
//      fifo_2 --> : : : : dp_sram <--> dum_ppu
//      cell_gen ba_cmu <--> dp_sram <--> dum_ppu
//      : : : : : : : : dp_sram <--> dum_ppu
//      : : : : : : : : dp_sram <--> dum_ppu
//
//      The cell_gen sends a variable traffic of cells
//      thru fifo_2, thru the ba_cmu, thru the dp_sram's,
//      to the dum_ppu's. The dum_ppu's then reformat
//      and resend the traffic back to the cell_gen,
//      which keeps track of cells, and flags cells that
//      haven't returned after its in-flight-cell buffer
//      overflows.
//
//      It's ok if a port times out, but you should be
//      able to keep it from happening by making the dum_ppus
//      more responsive (lower their pink noise dampening
//      factor).
//
// Synthesis Notes : Needs more development...
//
// Application Notes : Oh my, not now...
//
// Limitations      : N/A
//
// Bugs, Open issues, future enhancements :
//
//      make parameterized dp_sram_pro.v, adding rd_trash_cyc.
//      test rd_to_rd_wait = 0.
//      check read port scheduling fairness.
//      make sure wait states are followed, i.e. no bus contention.
//      test to see if unit recovers gracefully from exceptions.
//
// References       : Refer to the soon-to-be-completed docs in /Docs/...
//
// Disclaimer       : Copyright \2010 1999 Celox Communications Corporation
//                  All rights reserved
```

```
//
// Revision History : N/A
//
// $Log: ba_cmu_core.v,v $
// Revision 1.84 2000/07/12 21:59:14 celox
// Patched ba_cmu_core to fix OE timing bug.
//
// Revision 1.83 2000/04/07 12:58:50 acryan
// Changed output enable from 9 bits to 18 bits.
//
// Revision 1.82 2000/04/03 06:48:04 acryan
// Fixed small bug with indirect memory access and rd_to_wr_wait_cyc == 0.
//
// Revision 1.81 2000/03/29 10:37:59 acryan
// Fixed bug where pointers would not update after a cell transaction if
// port was shut down.
//
// Removed ptr_port_locked* variables.
//
// Revision 1.80 2000/03/28 16:52:01 acryan
// Fixed read scheduler bug from last commit.
//
// Fixed port shutdown and boot up design flaw. This manifested itself because
// the CMU didn't do a final update of the pointers before shutting down a port,
// causing cells to be lost and other cells to transmit twice.
//
// Revision 1.79 2000/03/27 13:17:50 acryan
// Fixed late FIFO sync lost on incomplete cell (not really a bug, an
// improvement).
//
// Revision 1.78 2000/03/27 11:42:54 acryan
// Fixed bug in which if a PPU updates the read tail pointer not on a cell
// boundary, and CMU gets to the end of the buffer, and there's no other
// read buffers to read from, the CMU will issue a FIFO sync lost interrupt
// on the next port that has a non-empty read buffer.
//
// The interrupt is still "late", but now it should issue the interrupt
// on the correct port. I may try to make the interrupt not "late" at
// another time.
//
// Revision 1.77 2000/03/27 10:33:38 acryan
// Made a few more changes for code coverage reasons.
//
// Removed impossible cases from code in port_dp (I hope).
//
// Revision 1.76 2000/03/24 13:02:16 acryan
// Removed one extraneous variable from a sensitivity list.
//
// Revision 1.75 2000/03/24 11:57:20 acryan
// Fixed bug so that if a write buffer overflows, the write tail shadow
// register is updated so that the PPU knows an (incomplete) cell has been
// written.
//
// Revision 1.74 2000/03/24 09:07:33 acryan
// Fixed bug where if two SOCCs enter the CMU, and the write buffers are
// full, the fifo_sync_lost interrupt will continually be issued.
//
// Split rd_gath_ctrl into two blocks for code coverage purposes.
//
// Revision 1.73 2000/03/23 12:38:55 acryan
// Fixed bug where CMU wrote to port 0 if all write buffers were full.
//
// It was harmless, unless the write buffers had overflowed, and then it
// erroneously issued the FIFO sync interrupt continuously.
//
// Revision 1.72 2000/03/22 18:57:23 acryan
// Fixed VCS simulation lockup bug. Works now
//
// Revision 1.71 2000/03/22 05:26:11 acryan
// Made array blocks into wires, and split write scheduler into two blocks.
//
// All for code coverage purposes
```

07/12/00
16:59:14

CMUcore_v1.0

2

```
// Revision 1.70 2000/03/16 05:39:52 acryan
// Fixed synth problem with internal regs and non-blocking statements.
//
// Revision 1.69 2000/03/14 11:18:48 acryan
// Made instantiation by name.
// Split ba_cm_addr into addr0, addr1.
//
// Revision 1.68 2000/02/19 10:50:15 acryan
// Very slight change in reset of new_port_req.
//
// Revision 1.67 2000/02/19 05:43:44 acryan
// Removed several critical paths by:
// 1. Flopping wr_buf_size.
// 2. Flopping rd_fifo_bias.
// 3. Flopping wr_oc_full, wr_oc_ov.
// 4. Tested with 1M cells.
//
// Revision 1.66 2000/02/14 07:12:46 acryan
// Fixed synthesis bug with CE/WE/OE and indexed indexes.
// Added comments to read_sched block.
//
// Revision 1.65 2000/02/11 14:18:39 acryan
// Fixed possible weird synthesis bug.
//
// Revision 1.64 2000/02/10 09:09:53 acryan
// Reduced code paths.
// Reduced some (hopefully) redundant logic.
// Made dum_mc do indirect memory access after every reboot of BA.
//
// Revision 1.63 2000/02/09 15:44:37 acryan
// Fixed totally obscure bug that happens only when the read tail and
// read head pointers wraparound and the occupancy becomes zero and
// ready latency is large and other conditions.
//
// Revision 1.62 2000/02/09 12:49:43 acryan
// The CMU now issues fifo_sync_lost interrupt in addition if:
// 1. An SOCC comes after another SOCC without a EOCC between.
// 2. If any data comes between an EOCC and SOCC.
// The write scheduler had to be re-written to support this.
// The CMU has been tested with 100K cells afterwards.
// The rd_new_port_dc is now 20 bits wide instead of 16 bits,
// to reduce the possibility of wrapping around on large cells.
// I renamed * next to * nxt.
// I renamed wr_scheduled to wr_sched_got_word.
// I added lots of comments in the code for clarity.
//
// Revision 1.61 2000/02/09 05:02:07 spatel
//
// Added Ambit directive for specifying synchronous reset signals.
//
// Revision 1.60 2000/02/08 15:04:07 acryan
// Got rid of most negedge statements in testbench.
// Unified DPRAM parity check.
// Split the write scheduler into two blocks.
//
// Revision 1.59 2000/02/08 10:49:30 acryan
// Made all testbench parameters into defines.
// Fixed a bug in write scheduler, wr_port should have been wr_port_next
// Fixed testbench bug where under light loads, cells would have large latencies.
// Fixed outgoing parity so it reflects bad parity from DPRAMs.
// Modified cell_gen so cells can have a timeout, like 1 ms.
//
// Revision 1.58 2000/02/07 11:14:44 acryan
// Fixed bug in read gatherer that would miss port change signals,
//
// which might cause it not to issue interrupts when it should.
// Also reduced the number of code paths, for coverage observability.
//
// Revision 1.57 2000/02/05 13:35:49 acryan
// Made CE_b and WE_b not do multiple non-blocking assignments.
//
// Revision 1.56 2000/02/05 12:30:53 acryan
// Changed ppu_dc to ppu_ic, making timeout param more responsive.
//
// Revision 1.55 2000/02/04 15:56:30 acryan
// Made rd_port_locked unlock much faster if port is CLOSED.
// Made port status scheduler more observable for code coverage.
//
// Revision 1.54 2000/02/04 13:33:27 acryan
// Fixed bug where after a premature SOCC from the read buffer would corrupt
// the next cell from the next port.
//
// Revision 1.53 2000/01/28 08:00:13 celox
// Fixed very minor bug with reloading and corrupted pointers
// corrupts the occupancy of the CLOSED port.
// Fixed very minor bug with ptr_port_misread now reset on reloading.
// Tested both code fixes.
//
// Revision 1.52 2000/01/28 02:58:18 celox
// Cosmetic changes to code.
//
// Revision 1.51 2000/01/25 06:17:32 celox
// Extended rd_word to be two cycles long, for timing closure purposes.
//
// Revision 1.50 2000/01/24 23:23:52 celox
// Just took out some display statements in combinational blocks.
//
// Revision 1.49 2000/01/16 04:03:47 celox
// Changed corrupted pointers from 2-bit to 3-bit saturating counter, should
// reduce false positives from every 1/4 second to every 1 year.
// Made read and write buffer occupancies reset on reset_b, for appearance's sake.
// Made CMU capable of handling single-word cells in both directions.
//
// Revision 1.48 2000/01/15 01:46:18 celox
// Did non-trivial rewrite of parts of the CMU to fix obscure bug that
// only manifested itself if there was:
// 1) long rd_latency.
// 2) cell shorter than rd_latency.
// 3) read buffer runs empty.
// 4) rd_port_dc is less than zero.
//
// Revision 1.47 2000/01/10 21:09:04 celox
// Fixed synthesizability problem with wr_word.
//
// Revision 1.46 2000/01/09 02:51:12 celox
// Removed lots of timescales.
// Also made CMU more observable for code coverage purposes.
//
// Revision 1.44 2000/01/08 03:15:26 celox
// Added rd_hd_ptr_shadow to fix bug.
// Fixed double pointer reload bug.
// Broke up seq blocks into smaller blocks for code coverage.
//
// Revision 1.43 1999/12/20 09:50:07 acryan
// Made test bench able to change config params on the fly and to
// reset the CMU occasionally. Fixed several reset-related bugs
// in the test bench and RTL.
//
// Revision 1.42 1999/12/17 16:25:19 acryan
// Implemented and tested indirect memory access.
```

07/12/00
16:59:14

ba cmu:core v. 1.0

3

```
// Revision 1.41 1999/12/17 06:45:23 acryan
// hi.
//
// Revision 1.40 1999/12/17 04:23:05 acryan
// Fixed potential bug of bus driver not delayed by wr_latency.
//
// Revision 1.39 1999/12/16 14:01:55 acryan
// Fixed some lint problems and commented out delay.
//
// Revision 1.38 1999/12/16 13:31:01 acryan
// Added indirect memory access infrastructure.
//
// Revision 1.37 1999/12/16 10:41:06 acryan
// Made a few minor changes for synthesis and code coverage purposes.
//
// Revision 1.36 1999/12/16 06:46:07 acryan
// Implemented the read scheduler to issue sync_lost interrupt, write
// partial cell, and set EOCC, if read buffer prematurely empty.
//
// Revision 1.35 1999/12/16 05:27:57 acryan
// Over 2M cells successfully tracked.
//
// Revision 1.34 1999/12/15 15:20:19 acryan
// 1. Implemented and tested HALTED ports able to become ACTIVE by receiving
// all cells, making write buffer occupancy zero.
// 2. Implemented HALTED ports' cells being redirected.
// 3. Implemented B bit packet continuity.
// 4. Implemented fifo sync lost logic, and the write scheduler's ability
// to handle and discard extra large packets.
// 5. Cleaned up the write scheduler code, reduced state.
// 6. Implemented write data state to force the data bus to low impedance.
// 7. Main FSM skips State 0 because of #6.
// 8. Renamed rd_signature config param to signature.
//
// Revision 1.33 1999/12/14 08:37:55 acryan
// Tested one-cycle reset_b, fixed possible bug with ppu timeout on a
// non-living port.
//
// Revision 1.32 1999/12/09 12:11:34 acryan
// Changed polarity of FIFO write and read signals in RTL and behavioral models.
//
// Revision 1.31 1999/12/09 10:07:25 acryan
// Changed slot_dc_eq_0 to slot_dc_lteq0, which should fix lots of scheduling
// glitches.
//
// Revision 1.30 1999/12/08 15:53:06 acryan
// Fixed some ExploreRTL problems.
//
// Revision 1.28 1999/12/04 13:18:28 acryan
// Added a more random port redirector using poly division.
//
// Revision 1.27 1999/12/04 07:31:44 acryan
// Removed 3 defunct nets.
//
// Revision 1.26 1999/12/01 08:44:47 acryan
// Changed ptr_s to ptr_shadow for clarity.
//
// Revision 1.25 1999/11/30 11:38:29 acryan
// Changed MAX_PORTS to NUM_PORTS and added some comments
//
// Revision 1.24 1999/11/30 06:33:02 acryan
// Removed stop statements from code using by verification team.
//
// Revision 1.23 1999/11/30 06:23:21 acryan
//
// Rewrote the read scheduler.
// Tightened bus timing.
// Fixed rd_to_rd_wait_cyc=0 bug.
//
// Revision 1.22 1999/11/29 12:58:36 acryan
// Fixed continuous fifo parity error error.
//
// Revision 1.21 1999/11/29 12:51:14 acryan
// Put anti-floating data bus measures into main FSM in CMU.
//
// Revision 1.20 1999/11/29 06:02:19 acryan
// Renamed reload signal to reload_req.
// Rewrote the read gatherer.
// Fixed rd_oc_div bug.
// Fixed use_mem_ce bug.
// Tested with 1M cells successfully.
//
// Revision 1.19 1999/11/26 12:16:30 acryan
// Registered FIFO oc signals, now CMU totally DFF isolated.
//
// Revision 1.18 1999/11/26 10:27:05 acryan
// Added a cycle of latency to FIFO occupancy and fixed CMU.
//
// Revision 1.17 1999/11/25 15:10:11 acryan
// Implemented and tested CMU "heartbeat". Changed port definitions on
// top-level CMU and top-level CIU.
//
// Revision 1.16 1999/11/25 11:09:52 acryan
// Made CMU dual-PPU SRAM capable.
//
// Revision 1.15 1999/11/23 12:41:49 acryan
// Fixed a bad rd_*_ptrs_gath bug.
//
// Revision 1.14 1999/11/22 17:05:45 acryan
// Made the test cell format more system compatible.
//
// Revision 1.13 1999/11/22 06:18:53 acryan
// Added a couple of comments.
//
// Revision 1.12 1999/11/19 09:21:37 acryan
// Rewrote CMU to make it more synthesizable. Also fixed ppu_timeout bug.
//
// Revision 1.12 1999/11/17 11:41:25 acryan
// *** empty log message ***
//
// Revision 1.11 1999/11/16 15:11:23 acryan
// When port halted, does not now redirect cells.
// Fixed cell_gen race conditions.
// Rerouted rd_signature and made BA now customizable thru MC.
// MC checks for interrupts now.
// Corrected the locations of the PPU and the CI bit in the ctrl cell format.
//
// Revision 1.10 1999/11/15 14:23:36 acryan
// Changed the pointer addresses to 0, 1, 16, 17.
//
// Revision 1.9 1999/11/15 11:43:23 acryan
// Changed CPU -> PPU, cleaned up port status code.
//
// Revision 1.8 1999/11/13 06:36:20 acryan
// Renamed wr_buf_size and rd_buf_size.
//
// Revision 1.7 1999/11/11 07:45:24 acryan
// Fixed the bug.
//
// Revision 1.6 1999/11/11 07:28:35 acryan
```

07/12/00
16:59:14

```
// Updated the parity from 8-bit to 16-bit. There are bugs in the code
// from the change, though.
//
// Revision 1.5 1999/11/06 06:50:22 acryan
// Removed timescale directive.
//
// Revision 1.4 1999/11/05 12:24:01 acryan
// Still debugging.
//
// Revision 1.3 1999/11/05 08:41:31 acryan
// Removed timescale directive.
//
// =====
//
// Inner-workings:
//
// There are six interdependent control blocks in the Control Memory Unit:
// Port status scheduler:
// Every cycle it checks the PPU timeout registers, and keeps
// each port in the correct status. This block is especially
// important when a PPU on a port needs to be reset or re-sync'ed
// up with the hardware.
// Main scheduler:
// This is the main finite state machine. It dictates when
// the "write" buffer is written, when the "read" buffer is read,
// and when the pointers are updated (read or written).
// Consequently, the main scheduler controls the remaining blocks.
// Read scheduler:
// It schedules which port gets to be read each cycle, and
// controls the read gatherer.
// Write scheduler:
// The write scheduler is not much of a scheduler, it just reads
// cells out of the FIFO, and routes the cell to the correct port.
// But it has spans 3 stages, and so is the most complex.
// Read gatherer:
// This block waits several cycles for the read data to get back
// from the external SRAM, and correctly stores the data.
// Pointers gatherer:
// This block is similar to the read gatherer, but it handles
// pointers that are read in off-chip.
//
```

```
'include "defines_ba_cmu.vh"

// ambit synthesis set_reset synchronous signals = "reset_b"

module ba_cmu_core (*AutoArg*)
// Outputs
CMU_CF3_data, CMU_CF3_prty, CMU_CF3_SOCC, CMU_CF3_BOCC,
CMU_CF3_write, CMU_CF2_read, BA_CM_data_out, BA_CM_prty_out,
BA_CM_BA_output, BA_CM_addr0, BA_CM_addr1, BA_CM_WE_b,
BA_CM_CE_b, BA_CM_OE_b, CMU_CIU_reload_ack, CMU_CIU_sync_lost,
CMU_CIU_timeout, CMU_CIU_corrupted_ptr, CMU_CIU_prty_err,
CMU_CIU_fifo_prty_err, CMU_CIU_fifo_sync_lost, CMU_CIU_frozen,
CMU_CIU_rd_oc0, CMU_CIU_rd_oc1, CMU_CIU_rd_oc2,
CMU_CIU_rd_oc3, CMU_CIU_wr_oc0, CMU_CIU_wr_oc1,
CMU_CIU_wr_oc2, CMU_CIU_wr_oc3, CMU_CIU_bus_util,
CMU_CIU_mem_data_out, CMU_CIU_mem_done,
// Inputs
clk, reset_b, CF3_CMU_oc, CF2_CMU_data, CF2_CMU_prty,
CF2_CMU_SOCC, CF2_CMU_BOCC, CF2_CMU_oc, CM_BA_data, CM_BA_prty,
CIU_CMU_ppu_timeout, rd_buf_size, SY_BA_wr_buf_size,
CIU_CMU_cyc_per_slot, CIU_CMU_cyc_per_entry,
```

```
CIU_CMU_wr_cyc_per_slot_min, CIU_CMU_wr_cyc_per_slot_max,
CIU_CMU_rd_port_oc_div, CIU_CMU_rd_port_oc_add, CIU_CMU_signature,
CIU_CMU_rd_to_rd_wait_cyc, CIU_CMU_wr_to_rd_wait_cyc,
CIU_CMU_rd_to_wr_wait_cyc, CIU_CMU_rd_latency, CIU_CMU_wr_latency,
CIU_CMU_use_mem_oe, CIU_CMU_use_mem_ce, CIU_CMU_dual_ppus,
CIU_CMU_fixed_redir_port_en, CIU_CMU_fixed_redir_port,
CIU_CMU_reload_req, CIU_CMU_mem_addr, CIU_CMU_mem_port,
CIU_CMU_mem_access, CIU_CMU_mem_ready_b, CIU_CMU_mem_write,
CIU_CMU_mem_data_in
);

input
input
clk;
reset_b;

// signals to and from Control FIFO 3 (write FIFO)
output [63:0] CMU_CF3_data;
output [3:0] CMU_CF3_prty;
output CMU_CF3_SOCC;
output CMU_CF3_BOCC;
output CMU_CF3_write;
input ['FIFO_ADDR-1:0] CF3_CMU_oc;

// signals to and from Control FIFO 2 (read FIFO)
input [63:0] CF2_CMU_data;
input [3:0] CF2_CMU_prty;
input CF2_CMU_SOCC;
input CF2_CMU_BOCC;
input CMU_CF2_read;
input ['FIFO_ADDR-1:0] CF2_CMU_oc;

// signals to and from the off-chip dual-ported SRAMS
input [63:0] CM_BA_data;
input [7:0] CM_BA_prty;
output [63:0] BA_CM_data_out;
output [7:0] BA_CM_prty_out;
output [17:0] BA_CM_BA_output;
output [19:0] BA_CM_addr0;
output [19:0] BA_CM_addr1;
output ['NUM_PORTS-1:0] BA_CM_WE_b;
output ['NUM_PORTS-1:0] BA_CM_CE_b;
output ['NUM_PORTS-1:0] BA_CM_OE_b;

// programmable registers
input [15:0] CIU_CMU_ppu_timeout;
input [3:0] rd_buf_size;
input [3:0] SY_BA_wr_buf_size;
input [7:0] CIU_CMU_cyc_per_slot;
input [1:0] CIU_CMU_cyc_per_entry;
input [7:0] CIU_CMU_wr_cyc_per_slot_min;
input [7:0] CIU_CMU_wr_cyc_per_slot_max;
input [3:0] CIU_CMU_rd_port_oc_div;
input [7:0] CIU_CMU_rd_port_oc_add;
input [7:0] CIU_CMU_signature;
input [2:0] CIU_CMU_rd_to_rd_wait_cyc;
input [2:0] CIU_CMU_wr_to_rd_wait_cyc;
input [2:0] CIU_CMU_rd_to_wr_wait_cyc;
input [2:0] CIU_CMU_rd_latency;
input [2:0] CIU_CMU_wr_latency;
input CIU_CMU_use_mem_oe;
input CIU_CMU_use_mem_ce;
input CIU_CMU_dual_ppus;
input CIU_CMU_fixed_redir_port_en;
input [1:0] CIU_CMU_fixed_redir_port;

// interrupts
```

07/12/00
16:59:14

ba_cmu_core.v

5

```
input {'NUM_PORTS-1:0} CMU_CIU_reload_req;
output {'NUM_PORTS-1:0} CMU_CIU_reload_ack;
output {'NUM_PORTS-1:0} CMU_CIU_sync_lost;
output {'NUM_PORTS-1:0} CMU_CIU_timeout;
output {'NUM_PORTS-1:0} CMU_CIU_corrupted_ptr;
output {'NUM_PORTS-1:0} CMU_CIU_prtty_err;
output CMU_CIU_fifo_prtty_err;
output CMU_CIU_fifo_sync_lost;
output CMU_CIU_frozen;

// status registers
output [19:0] CMU_CIU_rd_oc_0;
output [19:0] CMU_CIU_rd_oc_1;
output [19:0] CMU_CIU_rd_oc_2;
output [19:0] CMU_CIU_rd_oc_3;
output [19:0] CMU_CIU_wr_oc_0;
output [19:0] CMU_CIU_wr_oc_1;
output [19:0] CMU_CIU_wr_oc_2;
output [19:0] CMU_CIU_wr_oc_3;
output [15:0] CMU_CIU_bus_util;

// indirect memory access registers
input [19:0] CMU_CIU_mem_addr;
input [1:0] CMU_CIU_mem_port;
input CMU_CIU_mem_access;
input CMU_CIU_mem_ready_b;
input CMU_CIU_mem_write;
input [63:0] CMU_CIU_mem_data_in;
output [63:0] CMU_CIU_mem_data_out;
output CMU_CIU_mem_done;
reg [63:0] CMU_CIU_mem_data_out;
reg CMU_CIU_mem_done;

//////////////////////
// FSM states as parameters
//

parameter s0 = 0;
parameter s1 = 1;
parameter s2 = 2;
parameter s3 = 3;
parameter s4 = 4;
parameter s5 = 5;
parameter s6 = 6;
parameter s7 = 7;
parameter s8 = 8;
parameter s9 = 9;
parameter s10 = 10;
parameter s11 = 11;
parameter s12 = 12;

//////////////////////
// Datapath registers
//

// Interrupts:
reg {'NUM_PORTS-1:0} CMU_CIU_sync_lost; // out of sync interrupt
reg {'NUM_PORTS-1:0} CMU_CIU_sync_lost_nxt; // next value ...
reg {'NUM_PORTS-1:0} CMU_CIU_sync_lost_nxt_prelim; // .
reg {'NUM_PORTS-1:0} CMU_CIU_timeout; // timeout interrupt output
reg {'NUM_PORTS-1:0} CMU_CIU_corrupted_ptr; // corrupted ptr int
reg {'NUM_PORTS-1:0} CMU_CIU_corrupted_ptr_nxt; // ...
reg {'NUM_PORTS-1:0} CMU_CIU_prtty_err; // memory parity error int
```

```
reg CMU_CIU_fifo_prtty_err; // FIFO parity err int
reg CMU_CIU_fifo_prtty_err_nxt; // ...
reg CMU_CIU_fifo_sync_lost; // FIFO sync lost int
reg CMU_CIU_fifo_sync_lost_nxt; // ...
reg CMU_CIU_fifo_sync_lost_nxt_prelim; // ...
reg CMU_CIU_frozen;

// Stage WA:
reg CMU_CF2_read; // CF2 FIFO read signal
reg wr_fifo_read; // "next" value of CMU_CF2_read

wr_fifo_word; // data to be written to buffer
wr_fifo_word_P1; // "next" value of wr_fifo_word

CF2_CMU_oc_P1; // CF2 occupancy 1 cyc delay
CF3_CMU_oc_P1; // CF3 occupancy 1 cyc delay
we; // write enable, master
BA_output_nxt; // drive data enable
BA_CM_BA_output; // previous value of we
re; // read enable, master
re_port; // re demuxed by port
CMU_CIU_bus_util; // memory utilization metric
bus_util; // "next" value ...
bus_util_counter; // constantly incrementing cntnr
bus_util_counter_eq0; // bus_util_counter == 0
rd_addr_mask; // addr. mask (== buf_size-1)
wr_addr_mask; // ditto, except for write buf
wr_buf_size_P1; // prev value of wr_buf_size
rd_buf_size_P1; // prev value of wr_buf_size
wr_oc_full_thresh; // ditto, except for write buf
wr_oc_ov_thresh; // ditto, except for write buf
BA_CM_data_out; // wr_word delayed wr_latency
BA_CM_prtty_out; // wr_word delayed wr_latency
wr_word; // data to be written
wr_word_P1; // next value for wr_word
BA_CM_addr0; // memory address output copy 0
BA_CM_addr1; // memory address output copy 1
BA_CM_addr_nxt; // next memory address output
port; // memory port
port_nxt; // ...
port_rd_lat; // port delayed by rd_latency
port_rd_lat_P1; // "next" value of port_rd_lat
BA_CM_CE_b; // CE output for the 4 ports
BA_CM_OE_b; // OE output for the 4 ports
BA_CM_WE_b; // WE output for the 4 ports
wr_fifo_bias; // cf2_oc + cf3_oc
wr_cyc_per_slot; // write cycles per slot
wr_cyc_per_slot_nxt; // next value...
rd_cyc_per_slot; // cyc_per_slot-wr_cyc_per_slot
rd_cyc_per_slot; // dec. counter for r/w of slot
slot_dc; // slot_dc == 0
slot_dc_lteq0; // current ptr scheduler port
ptr_port; // ...
ptr_port_nxt; // ...
ptr_port_nxt_prelim; // ...
ptr_port_en; // whether ptr port is active
ptr_port_en_nxt; //
ptr_port_misread_0; // 2-bit sat counter for ptrs
ptr_port_misread_1; // 2-bit sat counter for ptrs
ptr_port_misread_2; // 2-bit sat counter for ptrs
ptr_port_misread_3; // 2-bit sat counter for ptrs
ptr_port_misread_nxt_0; // ...
ptr_port_misread_nxt_1; // ..
```

07/12/00
16:59:14

```
reg [2:0]
reg [1:0]
reg [1:0]
reg ['NUM_PORTS-1:0]
reg [8:0]
reg [15:0]
```

// Stage 1 Read:

```
reg
reg
reg
reg
reg [2:0]
reg [2:0]
reg [2:0]
reg [2:0]
reg
reg
reg
reg [3:0]
reg [3:0]
reg
reg
reg [1:0]
reg [1:0]
reg [19:0]
reg ['NUM_PORTS-1:0]
reg [19:0]
reg [19:0]
reg [19:0]
reg [19:0]
reg
reg
```

// Stage 1 Memory Access:

```
reg
reg
reg
reg
reg
reg
reg
```

// Stage 1 Write:

```
reg [1:0]
reg [1:0]
reg [1:0]
reg
reg
reg
reg
reg
reg
reg
reg
reg
reg [1:0]
reg [1:0]
reg [1:0]
reg ['NUM_PORTS-1:0]
reg ['NUM_PORTS-1:0]
```

```
ptr_port_misread_nxt_2; // ...
ptr_port_misread_nxt_3; // ...
rand_redir_port; // pseudo-random redirection port
ptr_port_updated; // used by INACTIVE/CLOSED ports
frozen_dc; // frozen counter
div_st; // redir_port random gen state
```

```
rd_re;
rd_re_rd_lat; // rd_re delayed by rd_latency
rd_re_rd_lat_f1; // "next" value ...
rd_re_rd_lat_f2; // "next" value ...
rd_to_rd_wait_dc; // dec. cntr of rd_to_rd_wait
rd_to_rd_wait_dc_nxt; // ...
rd_to_rd_wait_rd_dc; // used just for read sched
rd_to_rd_wait_rd_dc_nxt; // ...
rd_new_port_req; // req read sched for new port
rd_port_changed; // ack read gath for new port
rd_port_changed_nxt; // ...
rd_port_changed_dc; // ack read gath of new port
rd_port_changed_dc_nxt; // ...
rd_port_changed_rd_lat_f1; // "next" value
rd_port; // port of read scheduler
rd_port_nxt; // ...
rd_port_nxt_prelim; // ...
rd_new_port_dc; // dec. cntr for new read port
rd_new_port_dc_lt0; // rd_new_port_dc <= 0
rd_port_locked_nxt; // ...
rd_hd_ptr_rd_lat; // rd_hd_ptr delayed by rd_lat
rd_hd_ptr_rd_lat_f1; // "next" value ...
rd_oc_rd_lat; // rd_oc delayed by rd_latency
rd_oc_rd_lat_f1; // "next" value of rd_oc_rd_lat
rd_B_bit_set; // if B bit set in prev cell
rd_B_bit_set_nxt; // next value ...
```

```
mem_rd_gath; // memory access read
mem_rd_gath_rd_lat; // same delayed by rd_latency
mem_rd_gath_rd_lat_f1; // next value ...
mem_rd_gath_rd_lat_f2; // next value ...
mem_rd_done; // set once mem read is gath'd
mem_rd_data_out_hold_sel; // latches mem read
```

```
wr_sched_SOCC_state; // write sched state
wr_sched_SOCC_state_nxt; // next value ...
wr_sched_SOCC_state_nxt_prelim; //
wr_we; // write sched specific writ en
wr_we_prelim; //
wr_fifo_read_p1; // prev value of wr_fifo_read
wr_fifo_read_p1_f1; // prev value ...
wr_fifo_read_p2; // overflow val wr_fifo_read_p1
wr_fifo_read_p2_f1; // next value, OVERRUN/UNWIND
wr_fifo_word_ptrty_ok; // good par of wr_fifo_word
wr_fifo_SOCC; // start of control cell
wr_fifo_SOCC_F1; // next value, OVERRUN/UNWIND
wr_fifo_EOCC; // end of control cell
wr_fifo_EOCC_F1; // next value, OVERRUN/UNWIND
wr_port; // port used by write scheduler
wr_port_nxt; // next value of wr_port
wr_port_nxt_prelim; // next value of wr_port
wr_port_locked_nxt; // next value of ...
wr_port_locked_nxt_prelim; //
```

```
reg
// Stage 2+CIU_CMU_rd_latency
reg [71:0] rd_word_F1;
```

```
// Stage 3+CIU_CMU_rd_latency
reg [71:0] rd_word_FAN_TOP; // word read from read buffer
reg [71:0] rd_word_FAN_0A; // different fan out
reg [71:0] rd_word_FAN_0B; // ...
reg [71:0] rd_word_FAN_0C; // ...
reg [71:0] rd_word_FAN_1A; // ...
reg [71:0] rd_word_FAN_1B; // ...
reg [71:0] rd_word_FAN_1C; // ...
reg [71:0] rd_word_FAN_2A; // ...
reg [71:0] rd_word_FAN_2B; // ...
reg [71:0] rd_word_FAN_2C; // ...
reg [71:0] rd_word_FAN_3A; // ...
reg [71:0] rd_word_FAN_3B; // ...
reg [71:0] rd_word_FAN_3C; // ...
rd_word_eq_signature; // rd_word sig correct
rd_word_cell_len_eq1; // sched cell len equals 1
rd_word_ptrty_ok; // parity of rd_word correct?
rd_word_P1_eq_rd_word; // pointer read successful
rd_fifo_SOCC; // start of scheduled ctrl cell
rd_fifo_EOCC; // end of scheduled ctrl cell
rd_fifo_EOCC_prelim; // ...
rd_fifo_write; // "next" write enable for FIFO
```

```
// Stage 4+CIU_CMU_rd_latency
reg [63:0] CMU_CF3_data; // data to write FIFO
reg [3:0] CMU_CF3_ptrty; // parity to write FIFO
reg CMU_CF3_write; // write enable for FIFO
reg CMU_CF3_SOCC; // SOCC output to FIFO
reg CMU_CF3_EOCC; // EOCC output to FIFO
reg [71:0] rd_word_F1; // {CMU_CF3_data, parity}
reg [15:0] rd_cell_len_dc; // dec. cntr of read cell len
reg rd_cell_len_dc_eq0; // rd_cell_len_dc == 0
```

```
////////////////////////////////////
// Datapath control registers
//
```

```
// Stage WB:
reg wr_fifo_word_hold_sel; //
reg wr_fifo_word_load_sel; //

// Stage 1 Master:
reg [2:0] wr_word_sel; //
reg [2:0] addr_sel; //
reg [2:0] addr_sel_actual; // allows addr to eq SCRATCH
reg wr_cyc_per_slot_sel; //
reg rd_cyc_per_slot_sel; //
reg [1:0] slot_dc_sel; //
reg ppu_ic_reset_ptr; //
reg frozen_dc_reset; //
reg bus_util_sel; // true only for data transfers

// Stage 1 Read:
reg [1:0] rd_new_port_dc_sel; //
reg [1:0] rd_new_port_dc_sel_prelim; //
reg ['NUM_PORTS-1:0] rd_ptr_reloaded_sel; //
reg ['NUM_PORTS-1:0] rd_hd_ptr_rol sel; //
reg ['NUM_PORTS-1:0] rd_hd_ptr_load_sel; //
reg ['NUM_PORTS-1:0] rd_hd_ptr_hold_sel; //
```

```

reg ['NUM_PORTS-1:0] rd_oc_rollo_sel; //
reg ['NUM_PORTS-1:0] rd_oc_hold_sel; //
reg ['NUM_PORTS-1:0] rd_oc_hold_sel; //
reg ['NUM_PORTS-1:0] rd_hd_ptr_shadow_hold_sel; //
reg ['NUM_PORTS-1:0] rd_hd_ptr_shadow_load_sel; //

// Stage 1 Write:
reg [2:0] wr_word_sel_wr; //
wr_fifo_word_f1_hold_sel; //
wr_ptrs_reloaded_set_sel; //
reg ['NUM_PORTS-1:0] wr_hd_ptr_load_sel; //
reg ['NUM_PORTS-1:0] wr_tl_ptr_hold_sel; //
reg ['NUM_PORTS-1:0] wr_tl_ptr_load_sel; //
reg ['NUM_PORTS-1:0] wr_oc_hold_sel; //
reg ['NUM_PORTS-1:0] wr_oc_load_sel; //
reg ['NUM_PORTS-1:0] wr_tl_ptr_shadow_hold_sel; //
reg ['NUM_PORTS-1:0] wr_tl_ptr_shadow_load_sel; //

// Stage 3+CIU_CMU_rd_latency
reg [1:0] rd_cell_len_dc_sel; //

//////////////////////
// Inter-control registers
//

reg [3:0] master_state; // master FSM state
master_state_nxt; // ...
rd_to_wr_wait_dc; // ...
reg [3:0] rd_to_wr_wait_dc_nxt; //
wr_sched_finished; //
wr_sched_finished_prelim; //
wr_sched_go; //
wr_prev_word_moved; //
rd_sched_finished; //
rd_sched_go; //
wr_sched_go; //
rd_gath_state; //
reg [1:0] rd_gath_state_nxt; //
reg [1:0] rd_gath_state_nxt_prelim; //
rd_sched_state; //
rd_sched_state_nxt; //
rd_port_en; //
reg rd_rd_ptr_en_nxt; //
reg rd_rd_ptr_gath; //
reg rd_wr_ptrs_gath; //
reg rd_rd_ptr_gath_rd_lat; //
reg rd_rd_ptr_gath_rd_lat_f1; //
reg rd_rd_ptr_gath_rd_lat_f2; //
reg rd_wr_ptrs_gath_rd_lat; //
reg rd_wr_ptrs_gath_rd_lat_f1; //
reg rd_wr_ptrs_gath_rd_lat_f2; //

reg [1:0] redir_port;
reg [63:0] wr_word_data; // temp var used for wr_word
reg [15:0] bus_util_counter_nxt; // temp var for ...
reg ['NUM_PORTS-1:0] CE_b_nxt;
reg CE_tmp;
reg ['NUM_PORTS-1:0] WE_b_nxt;
reg WE_tmp;
reg ['NUM_PORTS-1:0] OE_b_nxt;
reg OE_tmp;
reg wr_cyc_per_slot_nxt_init; // temp var
reg wr_cyc_per_slot_underfl; // too small
reg wr_cyc_per_slot_overfl; // too big

reg ['NUM_PORTS-1:0] re_port_rd_lat_f1; // prev value ...
reg ['NUM_PORTS-1:0] re_port_rd_lat_f2; // prev value ...
reg ['NUM_PORTS-1:0] re_port_rd_lat_f3; // prev value ...
CF2_empty;
slot_dc_nxt; // temp var for slot_dc
rd_new_port_dc_nxt; // temp var for rd_new_port_dc
rd_cell_len_dc_nxt;

port_st_INDEX_ptr_port_nxt;
port_st_INDEX_wr_port_nxt;
port_st_INDEX_ptr_port;
port_st_INDEX_ptr_rd_lat;
ptr_port_misread_INDEX_ptr_rd_lat;
rd_oc_INDEX_ptr;
rd_hd_ptr_INDEX_ptr;
rd_hd_ptr_INDEX_rd_port_nxt;
rd_hd_ptr_shadow_INDEX_ptr_port_nxt;
wr_tl_ptr_INDEX_wr_port_nxt;
wr_tl_ptr_shadow_INDEX_ptr_port_nxt;

port_st_0; //
port_st_1; //
port_st_2; //
port_st_3; //
CMU_CIU_wr_oc_0; //
CMU_CIU_wr_oc_1; //
CMU_CIU_wr_oc_2; //
CMU_CIU_wr_oc_3; //
CMU_CIU_rd_oc_0; //
CMU_CIU_rd_oc_1; //
CMU_CIU_rd_oc_2; //
CMU_CIU_rd_oc_3; //
rd_hd_ptr_0; //
rd_hd_ptr_1; //
rd_hd_ptr_2; //
rd_hd_ptr_3; //
rd_hd_ptr_shadow_0; //
rd_hd_ptr_shadow_1; //
rd_hd_ptr_shadow_2; //
rd_hd_ptr_shadow_3; //
wr_tl_ptr_0; //
wr_tl_ptr_1; //
wr_tl_ptr_2; //
wr_tl_ptr_3; //
wr_tl_ptr_shadow_0; //
wr_tl_ptr_shadow_1; //
wr_tl_ptr_shadow_2; //
wr_tl_ptr_shadow_3; //
BA_CM_data_out_nxt; // ...
port_rd_lat_nxt; // ...
rd_hd_ptr_rd_lat_nxt; // next value of ..._F1
rd_oc_rd_lat_nxt; // next value of rd_oc_rd_lat_f1
re_port_rd_lat; // re_port delayed by rd_lat
BA_output_wr_lat_nxt; // delayed by wr_latency
mem_rd_gath_rd_lat_nxt; //

CMU_CIU_reload_ack; // reload ack per port
rd_port_locked; // track which port is read
rd_oc_eq0; // rd_oc[x] == 0
wr_port_locked; // tracks which port is written
wr_oc_eq0; // wr_oc[x] == 0
wr_oc_full; // wr_oc 3/4ths full
wr_oc_ov; // wr_oc overflows
rd_word_wr_hd_ptr_eq_wr_hd_ptr; // chk PPU halted
CMU_CIU_timeout_nxt; //

```

```

rd_port_nxt or
wr_port_nxt or
wr_we or
rd_re or
slot_dc_lteq0 or
CIU_CMU wr_to_rd_wait_cyc or
CIU_CMU rd_to_wr_wait_cyc or
rd_to_wr_wait_dc or
CIU_CMU rd_to_rd_wait_cyc or
rd_to_rd_wait_dc or
wr_word_sel_wr or
CIU_CMU mem_access or
CIU_CMU mem_ready_b or
CIU_CMU mem_write or
CIU_CMU mem_port or
port_st_INDEX_ptr_port or
ptr_port_nxt or
port_st_0 or
port_st_1 or
port_st_2 or
port_st_3 or
mem_rd_done
) begin: main_sched_ctrl

////////////////////
// Outputs
//
// Inter-control signals:
rd_wr_ptrs_gath = 0;
rd_rd_ptrs_gath = 0;

// Intra-control signals:
master_state_nxt = s0;
rd_to_wr_wait_dc_nxt = (rd_to_wr_wait_dc);
rd_to_rd_wait_dc_nxt = (rd_to_rd_wait_dc);

// Datapath control
addr_nxt = 7;
port_nxt = 0;
wr_word_sel = 0;
re = 0;
we = 0;
BA_output_nxt = 0;
wr_cyc_per_slot_sel = 1;
rd_cyc_per_slot_sel = 1;
slot_dc_sel = 0;
frozen_dc_reset = 0;
mem_rd_gath = 0;
CMU_CIU_mem_done = (mem_rd_done);

ptr_port_nxt = (ptr_port);
ptr_port_en_nxt = (ptr_port_en);
ptr_port_updated[ 'NUM_PORTS-1:0' ] = 0;

////////////////////
// Logic
//
case (master_state)
////////////////////

```


07/12/00
16:59:14

ba_cmu_core.v

```
// State 0: Pull data bus to ground
//
s0 : begin
  // Outputs
  wr_sched_go = 0;
  rd_sched_go = 0;
  BA_output_nxt = 1;
  wr_word_sel = 7;

  // keep data bus from floating
  // drive zeros

  // Compute next state
  master_state_nxt = s1;
end

// go to next state

////////////////////
// State 1: Write wr_ptrs 2, schedule R/W. ports
//
s1 : begin
  // Schedule buffer read/write
  wr_cyc_per_slot_sel = 0;
  rd_cyc_per_slot_sel = 0;
  wr_sched_go = 0;
  rd_sched_go = 0;

  // Schedule pointer ports
  (ptr_port_en_nxt, ptr_port_nxt_prelim) =
    (next_port (ptr_port,
      (port_st_0 != 'CLOSED'),
      (port_st_1 != 'CLOSED'),
      (port_st_2 != 'CLOSED'),
      (port_st_3 != 'CLOSED')));
  if (ptr_port_en_nxt) begin
    ptr_port_nxt = ptr_port_nxt_prelim;
    case (ptr_port_nxt)
      0 : port_st_INDEX_ptr_port_nxt = port_st_0;
      1 : port_st_INDEX_ptr_port_nxt = port_st_1;
      2 : port_st_INDEX_ptr_port_nxt = port_st_2;
      3 : port_st_INDEX_ptr_port_nxt = port_st_3;
    endcase
    // don't write pointers if port has RELOAD status
    we = (port_st_INDEX_ptr_port_nxt != 'RELOADING');
    end else begin
      we = 0;
    end

    // Outputs
    wr_word_sel = 2;
    addr_sel = 1;
    port_nxt = (ptr_port_nxt);
    BA_output_nxt = we;

    // Compute next state
    master_state_nxt = s2;
  // go to next state
end

////////////////////
// State 2: Write wr_ptrs 1
//
s2 : begin
  // Outputs
  slot_dc_sel = 3;
end

// hold wr_word
addr_sel = 0;
port_nxt = (ptr_port);
// don't write pointers if port has RELOAD status
we = (ptr_port_en && (port_st_INDEX_ptr_port != 'RELOADING));
BA_output_nxt = we;
wr_sched_go = 0;
rd_sched_go = 0;

// Compute next state
master_state_nxt = s3;
// skip over write buffer data
end

////////////////////
// State 3: Write buffer data (multi-cycle)
//
s3 : begin
  // Outputs
  wr_sched_go = 1;
  wr_word_sel = (wr_word_sel_wr);
  addr_sel = 5;
  port_nxt = (wr_port_nxt);
  we = (wr_we);
  BA_output_nxt = 1;
  frozen_dc_reset = 1;
  rd_sched_go = 0;

  // Compute next state
  if (wr_sched_finished || slot_dc_lteq0) begin
    if (CIU_CMU_wr_to_rd_wait_cyc) begin
      master_state_nxt = s4;
      // go to next state
    end else begin
      slot_dc_sel = 2;
      master_state_nxt = s5;
      // load rd_cyc_per_slot
      // skip over NOP to read bufs
    end else begin
      master_state_nxt = s3;
      // stay in same state
    end
  end

  // Compute next state
  if (wr_sched_finished || slot_dc_lteq0) begin
    if (CIU_CMU_wr_to_rd_wait_cyc) begin
      master_state_nxt = s4;
      // go to next state
    end else begin
      slot_dc_sel = 2;
      master_state_nxt = s5;
      // load rd_cyc_per_slot
      // skip over NOP to read bufs
    end
  end

  // State 4: NOP (optional, wr_to_rd_wait)
  //
s4 : begin
  // Outputs
  slot_dc_sel = 2;
  wr_sched_go = 0;
  rd_sched_go = 0;

  // Compute next state
  master_state_nxt = s5;
  // go to next state
end

////////////////////
// State 5: Read buffer data (multi-cycle)
//
s5 : begin
  // Outputs: read buffer data
  rd_sched_go = 1;
  addr_sel = 4;
end
```

07/12/00
16:59:14

```
port_nxt = (rd_port_nxt);
re = (rd_re);
wr_sched_go = 0;

// Compute next state
if (rd_sched_finished || slot_dc_lteq0) begin
    if (CIU_CMU_rd_to_rd_wait_cyc == 0) begin
        if (port_st_INDEX_ptr_port == 'RELOADING') begin
            master_state_nxt = s7; // read wr_ptrs
        end else begin
            master_state_nxt = s9; // skip over read wr_ptrs
        end
    end else begin
        rd_to_rd_wait_dc_nxt = (CIU_CMU_rd_to_rd_wait_cyc - 1);
        master_state_nxt = s6; // rd_to_rd_wait
    end
end else begin
    master_state_nxt = s5; // stay in same state
end // else: !if(rd_sched_finished || slot_dc_lteq0)

end // case: s5

////////////////////
// State 6: NOP (optional, multi-cycle, rd_to_rd_wait)
//
s6 : begin
    // Outputs
    wr_sched_go = 0;
    rd_sched_go = 0;

    // Compute next state
    if (rd_to_rd_wait_dc == 0) begin
        if (port_st_INDEX_ptr_port == 'RELOADING') begin
            master_state_nxt = s7; // read wr_ptrs
        end else begin
            master_state_nxt = s9; // skip over read wr_ptrs
        end
    end else begin
        rd_to_rd_wait_dc_nxt = (rd_to_rd_wait_dc - 1);
        master_state_nxt = s6; // stay in same state
    end
end

////////////////////
// State 7: Read wr_ptrs 2 (optional)
//
s7 : begin
    // Outputs: read wr_ptrs
    addr_sel = 1;
    port_nxt = (ptr_port);
    re = (ptr_port_en);
    wr_sched_go = 0;
    rd_sched_go = 0;

    // Compute next state
    master_state_nxt = s8, // go to next state
end

////////////////////
// State 8: Read wr_ptrs 1 (optional)
//
s8 : begin
    // Outputs: read wr_ptrs
    rd_wr_ptrs_gath = (ptr_port_en); // signal to pointers gather
    addr_sel = 0;
    port_nxt = (ptr_port);
    re = (ptr_port_en);
    wr_sched_go = 0;
    rd_sched_go = 0;

    // Compute next state
    master_state_nxt = s9; // go to next state
end

////////////////////
// State 9: Read rd_ptrs 2
//
s9 : begin
    // Outputs: read wr_ptrs
    addr_sel = 3;
    port_nxt = (ptr_port);
    re = (ptr_port_en);
    wr_sched_go = 0;
    rd_sched_go = 0;

    // Compute next state
    master_state_nxt = s10; // go to next state
end

////////////////////
// State 10: Read rd_ptrs 1
//
s10 : begin
    // Outputs: read wr_ptrs
    rd_rd_ptrs_gath = (ptr_port_en); // signal to pointers gather
    addr_sel = 2;
    port_nxt = (ptr_port);
    re = (ptr_port_en);
    wr_sched_go = 0;
    rd_sched_go = 0;

    // Set ptr_port_updated for INACTIVE/CLOSED port
    if (ptr_port_en) begin
        ptr_port_updated[ptr_port] = 1;
    end

    // Compute next state
    if (CIU_CMU_rd_to_wr_wait_cyc == 0) begin
        if (CIU_CMU_mem_access) begin
            rd_to_wr_wait_dc_nxt = 0;
            master_state_nxt = s11; // go to next state
        end else begin
            master_state_nxt = s1; // go to wr_ptrs state
        end
    end else begin
        rd_to_wr_wait_dc_nxt = (CIU_CMU_rd_to_wr_wait_cyc - 1);
        master_state_nxt = s11; // go to next state
    end
end
```

07/12/00
16:59:14

ba_cmucore.v

11

```
////////////////////
// State 11: NOP (optional, multi-cycle, rd_to_wr_wait)
//
s11 : begin
  // Outputs
  wr_sched_go = 0;
  rd_sched_go = 0;

  // Compute next state
  if (rd_to_wr_wait_dc == 0) begin
    if (CIU_CMU_mem_access) begin
      master_state_next = s12;
    end else begin
      master_state_next = s1;
    end
  end else begin
    rd_to_wr_wait_dc_next = (rd_to_wr_wait_dc - 1);
    master_state_next = s11;
  end
end

////////////////////
// State 12: Indirect memory access
//
s12 : begin
  // Outputs
  frozen_dc_reset = 1;
  wr_word_sel = 4;
  if (!CIU_CMU_mem_ready_b) begin
    addr_sel = 6;
    port_next = (CIU_CMU_mem_port);
    if (CIU_CMU_mem_write) begin
      we = 1;
      BA_output_next = 1;
      CMU_CIU_mem_done = 1;
    end else begin
      re = 1;
      mem_rd_gath = 1;
    end
  end else begin
    BA_output_next = 1;
    wr_sched_go = 0;
    rd_sched_go = 0;
    rd_to_wr_wait_dc_next = 11;
  end
  // wait a dozen cycles

  // Compute next state
  master_state_next = s11;
end

////////////////////
// Default: Make case statement full
//
default : begin
  wr_sched_go = 0;
  rd_sched_go = 0;
  master_state_next = s0;
end

////////////////////
endcase // case(master_state)
end // block: main_sched_ctrl

////////////////////
// Memory read gatherer
//
////////////////////
always @(mem_rd_gath_rd_lat or
port_rd_lat
) begin: mem_gath_ctrl
  // Outputs
  //
  // Datapath control
  mem_rd_data_out_hold_sel = 1;
  mem_rd_done = 0;

  // Logic
  //
  if (mem_rd_gath_rd_lat) begin
    mem_rd_data_out_hold_sel = 0;
    mem_rd_done = 1;
  end
end // block: mem_gath_ctrl

////////////////////
// FIFO gatherer
//
////////////////////
always @(posedge clk) begin: fifo_gath_seq
  if (!reset_b) begin
    wr_prev_word_moved <= 0;
  end else begin
    wr_prev_word_moved <= (wr_sched_got_word || 'wr_fifo_read_p2);
  end
end

always @(wr_sched_got_word or
wr_fifo_read_p2 or
cf2_empty or
wr_prev_word_moved
) begin: wr_prev_word_moved
  wr_prev_word_moved = 1;
end
```

1000

```

// The non-SOCC state is similar, except that no
// cells are ever discarded, unless there is an
// SOCC with a bad cell header signature. But
// all SOCCs in non-SOCC state cause a FIFO sync
// lost interrupt to be issued.
//
if (wr_sched_go) begin
  if (wr_fifo_read_P2) begin
    // if start of cell, try to schedule port
    if (wr_fifo_SOCC) begin
      // check signature field
      if (!wr_fifo_word_eq_signature) begin
        CMU_CIU_fifo_sync_lost_next_prelim = 1;
        end else begin
          // unlock all write ports
          wr_port_locked_next_prelim['NUM_PORTS-1:0'] = 0;
          // schedule port
          wr_port_next = (wr_fifo_word['CTRL_PPU']);
          case (wr_port_next)
            0 : port_st_INDEX wr_port_next = port_st_0;
            1 : port_st_INDEX wr_port_next = port_st_1;
            2 : port_st_INDEX wr_port_next = port_st_2;
            3 : port_st_INDEX wr_port_next = port_st_3;
          endcase // case(wr_port_next)
          if (wr_oc_full(wr_port_next) ||
              (port_st_INDEX wr_port_next != 'ACTIVE')) begin
            wr_word_sel_wr = 1; // set CI bit
            // One is subtracted from fixed_redir_port because
            // the function next_port gives the *next* available
            // port. To start with the current port, you have
            // to subtract one.
            redir_port = (CIU_CMU_fixed_redir_port_en ?
                          (CIU_CMU_fixed_redir_port - 1) :
                          rand_redir_port);
            (wr_sched_got_word, wr_port_next_prelim) =
              (next_port (redir_port,
                          (!wr_oc_full[0] &&
                           (port_st_0 == 'ACTIVE')),
                          (!wr_oc_full[1] &&
                           (port_st_1 == 'ACTIVE')),
                          (!wr_oc_full[2] &&
                           (port_st_2 == 'ACTIVE')),
                          (!wr_oc_full[3] &&
                           (port_st_3 == 'ACTIVE'))));
            wr_sched_finished_prelim = (!wr_sched_got_word);
            if (wr_sched_got_word) begin
              wr_port_next = wr_port_next_prelim;
            end
          end else begin // if (wr_oc_full(wr_port_next) || ...
            wr_sched_got_word = 1;
          end // else: !if (wr_oc_full(wr_port_next) || ...
            wr_we_prelim = wr_sched_got_word;
            if (wr_sched_got_word) begin
              // lock current write port
              wr_port_locked_next_prelim(wr_port_next) = 1;
              // go to non SOCC state
              wr_sched_SOCC_state_next_prelim = 0;
            end
          end // else: !if (wr_fifo_word_eq_signature)
          end else begin // if (wr_fifo_SOCC)
            wr_sched_got_word = 1;
            // issue interrupt if not SOCC at the start of cell,
            // discarding all words, else write the word
            if (wr_sched_SOCC_state) begin
              CMU_CIU_fifo_sync_lost_next_prelim = 1;
            end else begin
              wr_we_prelim = 1;
            end
          end // if (wr_fifo_SOCC)
          end // if (wr_fifo_read_P2)
        end // else: !if (wr_fifo_SOCC)
      end // if (wr_fifo_read_P2)
    end // block: wr_sched_ctrl_part1
  end
  always @(wr_sched_go or
            wr_fifo_read_P2 or
            wr_sched_got_word or
            wr_we_prelim or
            wr_port_locked_next_prelim or
            wr_sched_SOCC_state or
            wr_fifo_SOCC or
            wr_sched_finished_prelim or
            wr_oc_ov or
            wr_port_next or
            wr_fifo_BOCC or
            wr_fifo_word_prt_ok or
            CMU_CIU_fifo_sync_lost_next_prelim
            ) begin: wr_sched_ctrl_part2
    // Outputs
    //
    wr_we = (wr_we_prelim);
    wr_port_locked_next = (wr_port_locked_next_prelim);
    wr_tl_ptr_shadow_hold_sel['NUM_PORTS-1:0'] = ~(0);
    wr_sched_SOCC_state_next = (wr_sched_SOCC_state_next_prelim);
    wr_oc_hold_sel['NUM_PORTS-1:0'] = ~(0);
    wr_tl_ptr_hold_sel['NUM_PORTS-1:0'] = ~(0);
    wr_sched_finished = (wr_sched_finished_prelim);
    CMU_CIU_fifo_prt_err_next = 0;
    CMU_CIU_fifo_sync_lost_next = (CMU_CIU_fifo_sync_lost_next_prelim);
    // Logic
    //
    if (wr_sched_go) begin
      if (wr_fifo_read_P2) begin
        // if word is gotten, possibly write word
        if (wr_sched_got_word) begin
          if (wr_we) begin
            // if SOCC and not expecting SOCC, interrupt
            if (wr_fifo_SOCC && !wr_sched_SOCC_state) begin
              CMU_CIU_fifo_sync_lost_next = 1;
            end
            // if EOCC, unlock port, update shadow register,
            // wait for next SOCC.
            if (wr_fifo_BOCC) begin
              // unlock all write ports
              wr_port_locked_next['NUM_PORTS-1:0'] = 0;
              // load shadow register
              wr_tl_ptr_shadow_hold_sel(wr_port_next) = 0;
              wr_sched_SOCC_state_next = 1; // go to SOCC state
            end
          end
        end
      end
    end
  end
end

```

```

// if port overflows, issue interrupt, don't write,
// go into SOCC state so that the rest of the cell
// is discarded
if (wr_oc_ov[wr_port_nxt]) begin
    // unlock all write ports
    wr_port_locked_nxt['NUM_PORTS-1:0] = 0;
    // load shadow register
    wr_tl_ptr_shadow_hold_sel[wr_port_nxt] = 0;
    wr_sched_SOCC_state_nxt = 1; // go to SOCC state
    // issue interrupt
    CMU_CIU_fifo_sync_lost_nxt = 1;
    wr_we = 0;
end else begin
    // else update pointers
    wr_oc_hold_sel[wr_port_nxt] = 0;
    wr_tl_ptr_hold_sel[wr_port_nxt] = 0;
end

end // if (wr_we)
// check parity, and continue regardless
if (!wr_fifo_word_parity_ok) begin
    CMU_CIU_fifo_parity_err_nxt = 1;
end

end // if (wr_sched_got_word)
end else begin // if (wr_fifo_read_P2)
    // if no data to read, punt
    wr_sched_finished = 1;
    end // else: !if(wr_fifo_read_P2)
    end // if (wr_sched_go)
    end // block: wr_sched_ctrl_part2

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
// Read data scheduler
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

always @(posedge clk) begin: rd_sched_seq
    rd_to_rd_wait_rd_dc <= (rd_to_rd_wait_rd_dc_nxt);
    if (!reset_b) begin
        rd_sched_state <= s0;
        rd_port_changed <= 0;
        rd_port_changed_dc <= 0;
        rd_port_en <= 0;
    end else begin
        rd_sched_state <= (rd_sched_state_nxt);
        rd_port_changed <= (rd_port_changed_nxt);
        rd_port_changed_dc <= (rd_port_changed_dc_nxt);
        rd_port_en <= (rd_port_en_nxt);
    end
end // block: rd_sched_seq

always @(rd_sched_state or
rd_sched_go or
rd_port_en or
rd_port_locked or
rd_new_port_req or
rd_oc_eq0 or

```

```

CIU_CMU_rd_to_rd_wait_cyc or
rd_to_rd_wait_rd_dc or
CF3_CMU_oc_P1 or
rd_port_changed_dc or
port_st_0 or
port_st_1 or
port_st_2 or
port_st_3 or
CIU_CMU_rd_latency
) begin: rd_sched_ctrl

////////////////////////////////////
// Outputs
//
// Inter-control signals:
rd_sched_finished = 0;
rd_port_nxt = rd_port,
rd_re = 0;
rd_port_changed_nxt = 0;
rd_port_changed_dc_nxt = (rd_port_changed_dc == 0) ?
    0 : (rd_port_changed_dc - 1);

// Intra-control signals:
rd_sched_state_nxt = (rd_sched_state);
rd_to_rd_wait_rd_dc_nxt = (rd_to_rd_wait_rd_dc);
rd_port_en_nxt = (rd_port_en);

// Datapath control
rd_port_locked_nxt['NUM_PORTS-1:0] = rd_port_locked['NUM_PORTS-1:0];
rd_oc_hold_sel['NUM_PORTS-1:0] = ~(0);
rd_rd_ptr_hold_sel['NUM_PORTS-1:0] = ~(0);

////////////////////////////////////
// Logic
//
case (rd_sched_state)

////////////////////////////////////
// State 0: Read data
//
// Checks to see if FIFO 3 can handle more data. If so,
// it'll switch to a new port if the read gatherer
// requests it (because it found a cell boundary and
// it was time to change ports), or if the read buffer
// is empty, or if it was not able to schedule any ports
// to read the last time it tried. If it does find a
// port to read cells from, it first has to wait
// rd_to_rd_wait_cyc cycles before issuing read to
// the memory. If however, rd_to_rd_wait_cyc equals
// zero, and the port being switched to is the same as
// the current port, wait a cycle so that the rolling
// back of pointer and occupancy registers takes place
// before a read is issued.
//
s0 : begin
    if (rd_sched_go) begin
        // give 6 cycles lead time:
        // 1 for FIFO,
        // 1 for occupancy DFF

```

```

// 4 for internal stages
if (CF3_CMU_oc_P1 <
{'PIFO_SIZE - CIU_CMU_rd_latency - 7})
) begin
// Schedule next port
// if done gatherer requests new port, or
// if current rd oc empty, or
// if read scheduler is not doing anything
if (rd_new_port_req ||
rd_oc_eq0[rd_port] ||
!rd_port_en) begin
if (rd_port_en) begin
rd_port_changed_nxt = 1;
// disable new port reqs
rd_port_changed_dc_nxt = (CIU_CMU_rd_latency + 2);
end
// Unlock port
rd_port_locked_nxt['NUM_PORTS-1:0] = 0;
// Schedule next port
{rd_port_en_nxt, rd_port_nxt_prelim} =
(next_port (rd_port,
((port_st_0 == 'ACTIVE) ||
(port_st_0 == 'HALTED)) &&
!rd_oc_eq0[0]),
((port_st_1 == 'ACTIVE) ||
(port_st_1 == 'HALTED)) &&
!rd_oc_eq0[1]),
((port_st_2 == 'ACTIVE) ||
(port_st_2 == 'HALTED)) &&
!rd_oc_eq0[2]),
((port_st_3 == 'ACTIVE) ||
(port_st_3 == 'HALTED)) &&
!rd_oc_eq0[3])));
// Do rd_to_rd_wait if necessary
if (rd_port_en_nxt) begin
rd_port_nxt = rd_port_nxt_prelim;
rd_port_locked_nxt[rd_port_nxt] = 1;
if (CIU_CMU_rd_to_rd_wait_cyc == 0) begin
if (rd_port_nxt != rd_port) begin
// read data
rd_re = 1;
rd_oc_hold_sel[rd_port_nxt] = 0;
rd_hd_ptr_hold_sel[rd_port_nxt] = 0;
end
end else if (CIU_CMU_rd_to_rd_wait_cyc != 1) begin
rd_to_rd_wait_rd_dc_nxt =
(CIU_CMU_rd_to_rd_wait_cyc - 2);
rd_sched_state_nxt = sl; // go to rd_to_rd_wait
end
end else begin // if (rd_port_en_nxt)
rd_sched_finished = 1;
end // else: !if(rd_port_en_nxt)

end else begin // if (rd_new_port_req...
// if not scheduling another port, read data
rd_re = 1;
rd_oc_hold_sel[rd_port_nxt] = 0;
rd_hd_ptr_hold_sel[rd_port_nxt] = 0,

end // else: !if(rd_new_port_req ..

end else begin // if (CF3_CMU_oc_P1 < {'PIFO_SIZE...
rd_sched_finished = 1,
end // else: 'if(CF3_CMU_oc_P1 < {'PIFO_SIZE...

end // if (rd_sched_go)
end // case: s0

////////////////////
// State 1: rd_to_rd_wait
//
sl : begin
if (rd_to_rd_wait_rd_dc == 0) begin
rd_sched_state_nxt = s0; // Go to read state
end else begin
rd_to_rd_wait_rd_dc_nxt = (rd_to_rd_wait_rd_dc - 1);
end
end

endcase // case(rd_sched_state)
end // block: rd_sched_ctrl

////////////////////
// Read data gatherer
//
// Read data gatherer
//
////////////////////
always @(posedge clk) begin: rd_gath_seq
if (!reset_b) begin
rd_gath_state <= sl;
rd_new_port_req_P1 <= 0;
CMU_CF3_write <= 0;
CMU_CF3_SOCC <= 0;
CMU_CF3_EOCC <= 0;
rd_B_bit_set <= 0;
end else begin
rd_gath_state <= (rd_gath_state_nxt);
rd_new_port_req_P1 <= (rd_new_port_req);
CMU_CF3_write <= (rd_fifo_write);
CMU_CF3_SOCC <= (rd_fifo_SOCC);
CMU_CF3_EOCC <= (rd_fifo_EOCC);
rd_B_bit_set <= (rd_B_bit_set_nxt);
end
end // block: rd_gath_seq

always @(rd_gath_state or
rd_port_changed or
rd_re_rd_lat or
rd_word_eq_signature or
rd_cell_len_dc_eq0 or
rd_new_port_dc_lt0 or
port_rd_lat or
rd_new_port_req_P1 or
rd_port_changed_dc or
rd_B_bit_set or
rd_word_FAN_TOP or
port_st_INDEX_port_rd_lat or
rd_word_cell_len_eq1
) begin rd_gath_ctrl_part1

```

```

////////////////////
// Outputs
//
// Inter-control signals
rd_new_port_req = (rd_new_port_req_p1);

// Intra-control signals:
rd_gath_state_nxt_prelim = (rd_gath_state);
rd_B_bit_set_nxt = (rd_B_bit_set);

// Datapath control
rd_fifo_SOCC = 0;
rd_fifo_EOCC_prelim = 0;
rd_cell_len_dc_sel = 2; // hold rd_cell_len_dc
rd_new_port_dc_sel_prelim = 2; // hold rd_new_port_dc
rd_fifo_write = 0;

CMU_CIU_sync_lost_nxt_prelim('NUM_PORTS-1:0') = 0;
rd_hd_ptr_rol_sel('NUM_PORTS-1:0') = 0;
rd_oc_rol_sel('NUM_PORTS-1:0') = 0;
rd_hd_ptr_shadow_hold_sel('NUM_PORTS-1:0') = ~(0);

////////////////////
// Logic
//
// After much anguish, I choose the simpler, but
// slower scalar issue buffer reading. But this
// method still requires rolling back counter
// registers after false reads before a port
// switch.
//
// I wanted to be able to have more than one
// port in flight, i.e. while one port is finishing
// up and finding the end of a cell, the newest
// port would go ahead and start. Later, the
// the old port's counters would be rolled back.
// But the problem is that if the old and new
// ports are the same port, then the rollback
// doesn't effect the new "port", but it should.
// This would require much more logic and effort
// than I think its enhancements to the design
// deserve.
//
// if new port ack'ed, then turn off new port
if (rd_port_changed) begin
    rd_new_port_req = 0;
end

case (rd_gath_state)
////////////////////
// State 0: Discard read data
//
// Initial state. Used to throw away speculative reads.
// Can go into this state if the cell header signature
// is incorrect, or if it is time to switch read ports.
//
// Once the read scheduler signals that it has switched
// ports, the read gatherer goes to state 1 (SOC).

```

```

////////////////////
// State 1: Start of cell
//
// Checks cell header signature. Checks to see if the
// length of the cell is only one word. If so, checks to
// see if the port should be changed, either because
// rd_new_port decrementing counter has timed out, or
// the current port has deceased.
//
// If the read scheduler signals that is has switched
// ports, then no data should be thrown out, and the
// read gatherer goes to state 1 (SOC). If it is not
// EOCC, then a premature end of cell has occurred,
// so the read gathered issues a sync lost interrupt.
//
s1 : begin
    if (rd_re_rd_lat) begin
        rd_new_port_dc_sel_prelim = 0; // decrement rd_new_port_dc
    end
    // check signature
    if (!rd_word_eq_signature) begin
        CMU_CIU_sync_lost_nxt_prelim(port_rd_lat) = 1;
        if (rd_port_changed_dc == 0) begin
            rd_new_port_req = 1; // request port change
        end
        rd_gath_state_nxt_prelim = s0; // go to discard state
    end else begin
        rd_B_bit_set_nxt = rd_word_PAN_TOP['SCHED_B_BIT'];
        rd_cell_len_dc_sel = 1; // load
        rd_fifo_SOCC = 1; // set SOCC in FIFO
        rd_fifo_write = 1; // write to FIFO
        // check if EOCC
        if (rd_word_cell_len_eq1) begin
            rd_fifo_EOCC_prelim = 1; // set EOCC
            rd_hd_ptr_shadow_hold_sel(port_rd_lat) = 0;
            // If it's time to switch to a new port or the current
            // port is CLOSED, and it's allowed to switch to a new
            // port, and we are not inside a packet, switch ports.
            if ((rd_new_port_dc_lto ||
                ((port_st_INDEX_port_rd_lat != 'ACTIVE') &&
                 (port_st_INDEX_port_rd_lat != 'HALTED'))) &&
                (rd_port_changed_dc == 0) &&
                !rd_B_bit_set_nxt) begin
                rd_new_port_req = 1; // request port change
                // Roll back registers
                rd_hd_ptr_rol_sel(port_rd_lat) = 1;
                rd_oc_rol_sel(port_rd_lat) = 1;
                rd_gath_state_nxt_prelim = s0; // go to discard state
            end else begin
                rd_gath_state_nxt_prelim = s1; // go to SOC state
            end
        end else begin // if (rd_word_cell_len_eq1)
            rd_gath_state_nxt_prelim = s2, // go to next state
        end // else: !if(rd_word_cell_len_eq1)
        end // else: !if(!rd_word_eq_signature)
        end // if (rd_re_rd_lat)
    end // case s1

```


.....

07/12/00
16:59:14

```
//// Read pointers gatherer
////
////
always @(posedge clk) begin: ptr_gath_seq
  if (!reset_b) begin
    ptr_port_misread_0 <= 0;
    ptr_port_misread_1 <= 0;
    ptr_port_misread_2 <= 0;
    ptr_port_misread_3 <= 0;
  end else begin
    ptr_port_misread_0 <= ptr_port_misread_nxt_0;
    ptr_port_misread_1 <= ptr_port_misread_nxt_1;
    ptr_port_misread_2 <= ptr_port_misread_nxt_2;
    ptr_port_misread_3 <= ptr_port_misread_nxt_3;
  end
end // block: ptr_gath_seq

always @(rd_rd_ptr_gath_rd_lat or
         rd_wr_ptr_gath_rd_lat or
         rd_word_pl_eq_rd_word or
         port_rd_lat or
         rd_word_wr_hd_ptr_eq_wr_hd_ptr or
         port_st_INDEX_port_rd_lat or
         ptr_port_misread_INDEX_port_rd_lat or
         ptr_port_misread_0 or
         ptr_port_misread_1 or
         ptr_port_misread_2 or
         ptr_port_misread_3
        ) begin: ptr_gath_ctrl
  //////////////////////////////////////
  // Outputs
  //
  // Datapath control
  ptr_port_misread_nxt_0 = ptr_port_misread_0;
  ptr_port_misread_nxt_1 = ptr_port_misread_1;
  ptr_port_misread_nxt_2 = ptr_port_misread_2;
  ptr_port_misread_nxt_3 = ptr_port_misread_3;
  CMU_CIU_corrupted_ptr_nxt[NUM_PORTS-1:0] = 0;
  ppu_ic_reset_ptr[NUM_PORTS-1:0] = 0;
  rd_ptr_gath_reload_set_sel[NUM_PORTS-1:0] = 0;
  rd_hd_ptr_load_sel[NUM_PORTS-1:0] = 0;
  rd_oc_load_sel[NUM_PORTS-1:0] = 0;
  wr_ptr_gath_reload_set_sel[NUM_PORTS-1:0] = 0;
  wr_hd_ptr_load_sel[NUM_PORTS-1:0] = 0;
  wr_oc_load_sel[NUM_PORTS-1:0] = 0;
  rd_hd_ptr_shadow_load_sel[NUM_PORTS-1:0] = 0;
  wr_tl_ptr_shadow_load_sel[NUM_PORTS-1:0] = 0;
  //////////////////////////////////////
  // Logic
  //
  // gather normally written pointers
  if (rd_wr_ptr_gath_rd_lat) begin
    if (rd_word_pl_eq_rd_word) begin
```

```
      // load normally written out registers and update reload st
      // if read latency is large, make sure pointers are not
      // loaded twice, so check for port status.
      if (port_st_INDEX_port_rd_lat == 'RELOADING) begin
        // make sure the pointers don't get updated twice
        rd_hd_ptr_load_sel[port_rd_lat] = 1;
        wr_tl_ptr_shadow_load_sel[port_rd_lat] = 1;
        wr_ptr_gath_reload_set_sel[port_rd_lat] = 1;
        case (port_rd_lat)
          0 : ptr_port_misread_nxt_0 = 0;
          1 : ptr_port_misread_nxt_1 = 0;
          2 : ptr_port_misread_nxt_2 = 0;
          3 : ptr_port_misread_nxt_3 = 0;
        endcase
      end
      end else begin // if (rd_word_pl_eq_rd_word)
        // bail out of reload and retry
        CMU_CIU_corrupted_ptr_nxt[port_rd_lat] = 1;
        end // else: !if(rd_word_pl_eq_rd_word)

      end else begin // if (rd_wr_ptr_gath_rd_lat)
        // gather the read pointers
        if (rd_rd_ptr_gath_rd_lat) begin
          if (rd_word_pl_eq_rd_word) begin
            if (port_st_INDEX_port_rd_lat != 'CLOSED) begin
              // decrement misread counter
              if (ptr_port_misread_INDEX_port_rd_lat != 0) begin
                case (port_rd_lat)
                  0 : ptr_port_misread_nxt_0 = ptr_port_misread_0 - 1;
                  1 : ptr_port_misread_nxt_1 = ptr_port_misread_1 - 1;
                  2 : ptr_port_misread_nxt_2 = ptr_port_misread_2 - 1;
                  3 : ptr_port_misread_nxt_3 = ptr_port_misread_3 - 1;
                endcase // case(port_rd_lat)
              end
            end
          end
          // load occupancy registers and update reload status
          wr_hd_ptr_load_sel[port_rd_lat] = 1;
          rd_oc_load_sel[port_rd_lat] = 1;
          rd_ptr_gath_reload_set_sel[port_rd_lat] = 1;
          // reset ppu counter if write head pointer moved
          if ((rd_word_wr_hd_ptr_eq_wr_hd_ptr[port_rd_lat]) begin
            ppu_ic_reset_ptr[port_rd_lat] = 1;
          end
          end // if (port_st_INDEX_port_rd_lat != 'CLOSED)
        end else begin // if (rd_word_pl_eq_rd_word)
          if (port_st_INDEX_port_rd_lat == 'RELOADING) begin
            // bail out of reload and retry
            CMU_CIU_corrupted_ptr_nxt[port_rd_lat] = 1;
          end else begin
            // increment misread counter
            if (ptr_port_misread_INDEX_port_rd_lat == 7) begin
              CMU_CIU_corrupted_ptr_nxt[port_rd_lat] = 1;
            end else begin
              case (port_rd_lat)
                0 : ptr_port_misread_nxt_0 = ptr_port_misread_0 + 1;
                1 : ptr_port_misread_nxt_1 = ptr_port_misread_1 + 1;
                2 : ptr_port_misread_nxt_2 = ptr_port_misread_2 + 1;
                3 : ptr_port_misread_nxt_3 = ptr_port_misread_3 + 1;
              endcase
            end
          end
        end // else 'if(port_st_INDEX_port_rd_lat == 'RELOADING)
      end // else: 'if(rd_word_pl_eq_rd_word)
```

```

//
//
// Array structures
//
//

```

```

case (ptr_rd_lat)
  0 : ptr_port_musread_INDEX_ptr_rd_lat = ptr_port_musread_0;
  1 : ptr_port_musread_INDEX_ptr_rd_lat = ptr_port_musread_1;
  2 : ptr_port_musread_INDEX_ptr_rd_lat = ptr_port_musread_2;
  3 : ptr_port_musread_INDEX_ptr_rd_lat = ptr_port_musread_3;
endcase
end

```

```
always @ (rd_port_nxt or
rd_hd_ptr_0 or
rd_hd_ptr_1 or
rd_hd_ptr_2 or
rd_hd_ptr_3
```

```

) begin
  case (rd_port_nxt)
    0 : rd_hd_ptr_INDEX_rd_port_nxt = rd_hd_ptr_0;
    1 : rd_hd_ptr_INDEX_rd_port_nxt = rd_hd_ptr_1;
    2 : rd_hd_ptr_INDEX_rd_port_nxt = rd_hd_ptr_2;
    3 : rd_hd_ptr_INDEX_rd_port_nxt = rd_hd_ptr_3;
  endcase
end

always @(wr_port_nxt or
         wr_tl_ptr_0 or
         wr_tl_ptr_1 or
         wr_tl_ptr_2 or
         wr_tl_ptr_3
        ) begin
  case (wr_port_nxt)
    0 : wr_tl_ptr_INDEX_wr_port_nxt = wr_tl_ptr_0;
    1 : wr_tl_ptr_INDEX_wr_port_nxt = wr_tl_ptr_1;
    2 : wr_tl_ptr_INDEX_wr_port_nxt = wr_tl_ptr_2;
    3 : wr_tl_ptr_INDEX_wr_port_nxt = wr_tl_ptr_3;
  endcase
end

//////////
// Control signals delayed by CIU_CMU_rd_latency
// (or CIU_CMU_wr_latency)
//
always @(port_nxt or
         re
        ) begin
  re_port[\'NUM_PORTS-1:0] = 0;
  if (re) begin
    re_port[port_nxt] = 1;
  end
end

ba_cmu_var_delay ba_cmu_var_delay_BA_output
(
  .clk (clk),
  .reset_b (1'b1),
  .in (BA_output_nxt),
  .delay (CIU_CMU_wr_latency),
  .out (BA_output_wr_lat_nxt)
);

ba_cmu_var_delay #(72) ba_cmu_var_delay_wr_word
(
  .clk (clk),
  .reset_b (1'b1),
  .in (wr_word),
  .delay (CIU_CMU_wr_latency),
  .out (BA_CM_data_out_nxt)
);

ba_cmu_var_delay #(\'NUM_PORTS) ba_cmu_var_delay_re_port
(
  .clk (clk),
  .reset_b (1'b1),
  .in (re_port),
  .delay (CIU_CMU_rd_latency),

```

```

  .out (re_port_rd_lat)
);

ba_cmu_var_delay #(1, \'OR_PREV_DELAY) ba_cmu_var_delay_re_spread_prev
(
  .clk (clk),
  .reset_b (1'b1),
  .in (re),
  .delay (CIU_CMU_rd_latency),
  .out (re_rd_lat_spread_prev)
);

ba_cmu_var_delay #(2) ba_cmu_var_delay_port
(
  .clk (clk),
  .reset_b (1'b1),
  .in (port),
  .delay (CIU_CMU_rd_latency),
  .out (port_rd_lat_nxt)
);

ba_cmu_var_delay #(20) ba_cmu_var_delay_rd_oc
(
  .clk (clk),
  .reset_b (1'b1),
  .in (rd_oc_INDEX_port),
  .delay (CIU_CMU_rd_latency),
  .out (rd_oc_rd_lat_nxt)
);

ba_cmu_var_delay #(20) ba_cmu_var_delay_rd_hd_ptr
(
  .clk (clk),
  .reset_b (1'b1),
  .in (rd_hd_ptr_INDEX_port),
  .delay (CIU_CMU_rd_latency),
  .out (rd_hd_ptr_rd_lat_nxt)
);

ba_cmu_var_delay ba_cmu_var_delay_rd_re
(
  .clk (clk),
  .reset_b (reset_b),
  .in (rd_re),
  .delay (CIU_CMU_rd_latency),
  .out (rd_re_rd_lat_nxt)
);

ba_cmu_var_delay ba_cmu_var_delay_rd_port_changed
(
  .clk (clk),
  .reset_b (reset_b),
  .in (rd_port_changed_nxt),
  .delay (CIU_CMU_rd_latency),
  .out (rd_port_changed_rd_lat_nxt)
);

ba_cmu_var_delay ba_cmu_var_delay_rd_rd_ptrs_gath
(
  .clk (clk),
  .reset_b (reset_b),
  .in (rd_rd_ptrs_gath),
  .delay (CIU_CMU_rd_latency),
  .out (rd_rd_ptrs_gath_rd_lat_nxt)
);

```

```

ba_cmu_var_delay  ba_cmu_var_delay_rd_wr_ptrsr_gath
(
    .clk (clk),
    .reset_b (reset_b),
    .in (rd_wr_ptrsr_gath),
    .delay (CIU_CMU_rd_latency),
    .out (rd_wr_ptrsr_gath_rd_lat_nxt)
);

ba_cmu_var_delay  ba_cmu_var_delay_mem_rd_gath
(
    .clk (clk),
    .reset_b (reset_b),
    .in (mem_rd_gath),
    .delay (CIU_CMU_rd_latency),
    .out (mem_rd_gath_rd_lat_nxt)
);

always @(posedge clk) begin: var_delay_seq
    BA_CM_BA_output <= {(18(BA_output_wr_lat_nxt))};
    BA_CM_data_out <= BA_CM_data_out_nxt[63:0];
    BA_CM_ptrty_out <= BA_CM_data_out_nxt[71:64];
    // re_port_rd_lat is not registered
    // re_rd_lat_spread_prev is not registered
    port_rd_lat_F1 <= port_rd_lat_nxt;    // parallel to rd_word
    port_rd_lat <= port_rd_lat_F1;
    rd_oc_rd_lat_F1 <= rd_oc_rd_lat_nxt;    // parallel to rd_word
    rd_oc_rd_lat <= rd_oc_rd_lat_F1;
    rd_hd_ptr_rd_lat_F1 <= rd_hd_ptr_rd_lat_nxt; // parallel to rd_word
    rd_hd_ptr_rd_lat <= rd_hd_ptr_rd_lat_F1;
    if (!reset_b) begin
        re_port_rd_lat_F1 <= 0;    // parallel to rd_word
        re_port_rd_lat_P2 <= 0;
        re_port_rd_lat_P3 <= 0;
        rd_re_rd_lat_F2 <= 0;
        rd_re_rd_lat_F1 <= 0;
        rd_re_rd_lat <= 0;
        rd_rd_ptrsr_gath_rd_lat_F2 <= 0;    // parallel to rd_word
        rd_rd_ptrsr_gath_rd_lat_F1 <= 0;
        rd_rd_ptrsr_gath_rd_lat <= 0;
        rd_wr_ptrsr_gath_rd_lat_F2 <= 0;    // parallel to rd_word
        rd_wr_ptrsr_gath_rd_lat_F1 <= 0;
        rd_wr_ptrsr_gath_rd_lat <= 0;
        mem_rd_gath_rd_lat_F2 <= 0;
        mem_rd_gath_rd_lat_F1 <= 0;
        mem_rd_gath_rd_lat <= 0;
        rd_port_changed_rd_lat_F1 <= 0;    // parallel to rd_word
        rd_port_changed_rd_lat <= 0;
    end else begin
        re_port_rd_lat_P1 <= re_port_rd_lat;
        re_port_rd_lat_P2 <= re_port_rd_lat_P1;
        re_port_rd_lat_P3 <= re_port_rd_lat_P2;
        rd_re_rd_lat_F2 <= rd_re_rd_lat_nxt;
        rd_re_rd_lat_F1 <= rd_re_rd_lat_F2;
        rd_re_rd_lat <= rd_re_rd_lat_F1;

```

```

rd_rd_ptrsr_gath_rd_lat_F2 <= rd_rd_ptrsr_gath_rd_lat_nxt;
rd_rd_ptrsr_gath_rd_lat_F1 <= rd_rd_ptrsr_gath_rd_lat_F2;
rd_rd_ptrsr_gath_rd_lat <= rd_rd_ptrsr_gath_rd_lat_F1;
rd_wr_ptrsr_gath_rd_lat_F2 <= rd_wr_ptrsr_gath_rd_lat_nxt;
rd_wr_ptrsr_gath_rd_lat_F1 <= rd_wr_ptrsr_gath_rd_lat_F2;
rd_wr_ptrsr_gath_rd_lat <= rd_wr_ptrsr_gath_rd_lat_F1;
mem_rd_gath_rd_lat_F2 <= mem_rd_gath_rd_lat_nxt;
mem_rd_gath_rd_lat_F1 <= mem_rd_gath_rd_lat_F2;
mem_rd_gath_rd_lat <= mem_rd_gath_rd_lat_F1;
rd_port_changed_rd_lat_F1 <= rd_port_changed_rd_lat_nxt;
rd_port_changed_rd_lat <= rd_port_changed_rd_lat_F1;
end // else: !if(!reset_b)
end // block: var_delay_seq

////////////////////
// Frequently used priority encoding scheduler
//

function [2:0] next_port;
input [1:0] current_port;
input port_st_0, port_st_1, port_st_2, port_st_3;

case ({current_port, port_st_0, port_st_1, port_st_2, port_st_3})
    6'b00_0100 : next_port = 3'b101;
    6'b00_0101 : next_port = 3'b101;
    6'b00_0110 : next_port = 3'b101;
    6'b00_0111 : next_port = 3'b101;
    6'b00_1100 : next_port = 3'b101;
    6'b00_1101 : next_port = 3'b101;
    6'b00_1110 : next_port = 3'b101;
    6'b00_1111 : next_port = 3'b101;

    6'b00_0010 : next_port = 3'b110;
    6'b00_0011 : next_port = 3'b110;
    6'b00_1010 : next_port = 3'b110;
    6'b00_1011 : next_port = 3'b110;

    6'b00_0001 : next_port = 3'b111;
    6'b00_1001 : next_port = 3'b111;

    6'b00_1000 : next_port = 3'b100;
    6'b00_0000 : next_port = 3'b000;    // don't care, 3'b00xx

    6'b01_0010 : next_port = 3'b110;
    6'b01_0011 : next_port = 3'b110;
    6'b01_0110 : next_port = 3'b110;
    6'b01_0111 : next_port = 3'b110;
    6'b01_1010 : next_port = 3'b110;
    6'b01_1011 : next_port = 3'b110;
    6'b01_1110 : next_port = 3'b110;
    6'b01_1111 : next_port = 3'b110;

    6'b01_0001 : next_port = 3'b111;
    6'b01_0101 : next_port = 3'b111;
    6'b01_1001 : next_port = 3'b111;
    6'b01_1101 : next_port = 3'b111;

    6'b01_1000 : next_port = 3'b100;
    6'b01_1100 : next_port = 3'b100;

```

```

6'b01_0100 : next_port = 3'b101;
6'b01_0000 : next_port = 3'b000;

// don't care, 3'b0xx

6'b10_0001 : next_port = 3'b111;
6'b10_0011 : next_port = 3'b111;
6'b10_0101 : next_port = 3'b111;
6'b10_0111 : next_port = 3'b111;
6'b10_1001 : next_port = 3'b111;
6'b10_1011 : next_port = 3'b111;
6'b10_1101 : next_port = 3'b111;
6'b10_1111 : next_port = 3'b111;

6'b10_1000 : next_port = 3'b100;
6'b10_1010 : next_port = 3'b100;
6'b10_1100 : next_port = 3'b100;
6'b10_1110 : next_port = 3'b100;

6'b10_0100 : next_port = 3'b101;
6'b10_0110 : next_port = 3'b101;

6'b10_0010 : next_port = 3'b110;
6'b10_0000 : next_port = 3'b000;

// don't care, 3'b0xx

6'b11_1000 : next_port = 3'b100;
6'b11_1001 : next_port = 3'b100;
6'b11_1010 : next_port = 3'b100;
6'b11_1011 : next_port = 3'b100;
6'b11_1100 : next_port = 3'b100;
6'b11_1101 : next_port = 3'b100;
6'b11_1110 : next_port = 3'b100;
6'b11_1111 : next_port = 3'b100;

6'b11_0100 : next_port = 3'b101;
6'b11_0101 : next_port = 3'b101;
6'b11_0110 : next_port = 3'b101;
6'b11_0111 : next_port = 3'b101;

6'b11_0010 : next_port = 3'b110;
6'b11_0011 : next_port = 3'b110;

6'b11_0001 : next_port = 3'b111;
6'b11_0000 : next_port = 3'b000;

// don't care, 3'b0xx

endcase // case((current_port, port_st_0, port_st_1, port_st_2, port_st_3))

endfunction // next_port

////////////////////////////////////
// Parity checker
//

function prty_ok_68;
input {67:0} word;

if ((word[67] != (~word[63:56])) ||
(word[66] != (~word[47:32])) ||
(word[65] != (~word[31:16])) ||
(word[64] != (~word[15: 0]))) begin
prty_ok_68 = 0;
end else begin

prty_ok_68 = 1;
end

endfunction // prty_ok_68

function prty_ok_72;
input {71:0} word;

if ((word[71] != (~word[63:56])) ||
(word[70] != (~word[55:48])) ||
(word[69] != (~word[47:40])) ||
(word[68] != (~word[39:32])) ||
(word[67] != (~word[31:24])) ||
(word[66] != (~word[23:16])) ||
(word[65] != (~word[15: 8])) ||
(word[64] != (~word[ 7: 0]))) begin
prty_ok_72 = 0;
end else begin

prty_ok_72 = 1;
end // else: !if((word[71] != (~word[63:56])) || ...

endfunction // prty_ok_72

////////////////////////////////////
// Signal directory (and which block controls it):
//
// Interrupts:
// CMU_CIU_sync_lost[3:0] // read gatherer
// CMU_CIU_timeout[3:0] // port status scheduler
// CMU_CIU_corrupted_ptr[3:0] // pointer gatherer
// CMU_CIU_prty_err[3:0] // read gatherer, pointer gatherer
// CMU_CIU_fifo_prty_err // write scheduler
//
// Stage WA:
//
// Stage WB:
// wr_ptr_ptr_P1[0:3] // write scheduler
// wr_fifo_word // write scheduler
// wr_fifo_SOCC // write scheduler
// wr_fifo_EOCC // write scheduler
//
// Stage 1:
// Master:
// wr_word // main scheduler
// addr // main scheduler
// port // main scheduler
// BA_CM_CE_b[0:3] // main scheduler
// BA_CM_OE_b[0:3] // main scheduler
// BA_CM_WE_b[0:3] // main scheduler
// rd_cyc_per_slot // main scheduler
// wr_cyc_per_slot // main scheduler
// slot_dc // main scheduler
// ptr_port // main scheduler

```

```

// ptr_port_en // main scheduler
// rand_redir_port // (no select)
// port_st[0:3] // port status scheduler
// reload_ack[0:3] // port status scheduler
// ppu_ic[0:3] // pointers gatherer

Read:
// rd_port // read scheduler
// rd_new_port_dc // read scheduler
// rd_port_locked[0:3] // read scheduler
// rd_ptrs_reloaded[0:3] // pointers gatherer
// rd_hd_ptr[0:3] // (ptrs gath, read gath, rd sch)
// rd_oc[0:3] // (ptrs gath, read gath, rd sch)

Write:
// wr_port_locked[0:3] // write scheduler
// wr_ptrs_reloaded[0:3] // rd wr_ptrs gatherer
// wr_tl_ptr[0:3] // (pointers gatherer, write sch)
// wr_oc[0:3] // (pointers gatherer, write sch)
// wr_tl_ptr_shadow[0:3] // (pointers gatherer, write sch)
// wr_port // write scheduler
// wr_oc_full[0:3] (assigned)

Stage 2:
Read:
// rd_oc_eq0[0:3] (assigned)

Stage 2+CIU_CMU_rd_latency:
Read:
// rd_word // (no select)

Stage 3+CIU_CMU_rd_latency:
Read:
// CMU_CF3_data, CMU_CF3_ptrty (rd_word_P1) // (no select)
// rd_cell_len_dc // read gatherer
// rd_cell_len_dc_eq0 (assigned)

//
//
// CF2_empty
//
always @(CF2_CMU_oc_P1 or
wr_fifo_read_P1 or
wr_fifo_read_P1_P1) begin
CF2_empty = ((CF2_CMU_oc_P1 == 0) ||
(CF2_CMU_oc_P1 == 1) &&
(wr_fifo_read_P1 || wr_fifo_read_P1_P1) ||
(CF2_CMU_oc_P1 == 2) &&
wr_fifo_read_P1 && wr_fifo_read_P1_P1));

end

//
//
// wr_word
//
// 0 -- Data from FIFO 2
// 1 -- Data from FIFO 2, with CI bit set
// 2 -- CMU-controlled pointers
// 3 -- Previous value of wr_word
// 4, 5 -- Indirect memory access data
// 6, 7 -- Zeros

always @(wr_word_sel or
wr_fifo_word or
wr_word_P1 or

```

```

rd_hd_ptr_shadow_INDEX_ptr_port_nxt or
wr_tl_ptr_shadow_INDEX_ptr_port_nxt or
CIU_CMU_mem_data_in
) begin: wr_word_block

case (wr_word_sel)
0 : wr_word_data = (wr_fifo_word[63:0]);
1 : wr_word_data = ({wr_fifo_word[63:'CTRL_CI_BIT+1], 1'b1,
wr_fifo_word['CTRL_CI_BIT-1:0]});
2 : wr_word_data = ({12'b0, rd_hd_ptr_shadow_INDEX_ptr_port_nxt,
12'b0, wr_tl_ptr_shadow_INDEX_ptr_port_nxt});
3 : wr_word_data = (wr_word_P1[63:0]);
4, 5 : wr_word_data = (CIU_CMU_mem_data_in);
6, 7 : wr_word_data = 64'h0000_0000_0000_0000;
endcase

wr_word = ({(!^wr_word_data[63:56]),
!/^wr_word_data[55:48]),
!/^wr_word_data[47:40]),
!/^wr_word_data[39:32]),
!/^wr_word_data[31:24]),
!/^wr_word_data[23:16]),
!/^wr_word_data[15: 8]),
!/^wr_word_data[ 7: 0]),
wr_word_data});

end

//
// wr_fifo_word_ptrty_ok
//
always @(wr_fifo_word) begin
wr_fifo_word_ptrty_ok = (ptrty_ok_68 (wr_fifo_word));
end

//
// wr_fifo_word_eq_signature
//
always @(wr_fifo_word or
CIU_CMU_signature
) begin
wr_fifo_word_eq_signature =
(wr_fifo_word['CTRL_SIGNATURE] == CIU_CMU_signature);
end

//
// rd_word_P1_eq_rd_word
//
always @(rd_word_P1 or rd_word_FAN_TOP) begin
rd_word_P1_eq_rd_word = (rd_word_P1 == rd_word_FAN_TOP);
end

//
// rd_word_eq_signature
//
always @(rd_word_FAN_TOP or
CIU_CMU_signature
) begin

```

```

rd_word_eq_signature = (rd_word_FAN_TOP['SCHED_SIGNATURE'] ==
    CIU_CMU_signature);
end

//////////
// rd_word_cell_len_eq1
//
always @(rd_word_FAN_TOP) begin
    rd_word_cell_len_eq1 = (rd_word_FAN_TOP['SCHED_CELL_LEN'] == 1);
end

//////////
// rd_word_prty_ok
//
always @(rd_word_FAN_TOP) begin
    rd_word_prty_ok = (prty_ok_72 (rd_word_FAN_TOP));
end

//////////
// Interrupts
//
//////////
//
//////////

always @(posedge clk) begin
    CMU_CIU_sync_lost['NUM_PORTS-1:0'] <=
        (CMU_CIU_sync_lost_nxt['NUM_PORTS-1:0']);
    CMU_CIU_timeout['NUM_PORTS-1:0'] <=
        (CMU_CIU_timeout_nxt['NUM_PORTS-1:0']);
    CMU_CIU_corrupted_ptr['NUM_PORTS-1:0'] <=
        (CMU_CIU_corrupted_ptr_nxt['NUM_PORTS-1:0']);
    (re_port_rd_lat_p3 & {'NUM_PORTS'(rd_word_prty_ok)});
    CMU_CIU_fifo_prty_err <= (CMU_CIU_fifo_prty_err_nxt);
    CMU_CIU_fifo_sync_lost <= (CMU_CIU_fifo_sync_lost_nxt);
end

//////////
//
//////////
// Stage WB
//
//////////
//
//////////
// rd_addr_mask
//
always @(posedge clk) begin
    rd_buf_size_P1 <= (rd_buf_size),

```

```

end

always @(posedge clk) begin
    case (rd_buf_size_P1)
        0 : rd_addr_mask <= 20'h0000ff; // 16 words == 128 B
        1 : rd_addr_mask <= 20'h0001ff; // 32 words == 256 B
        2 : rd_addr_mask <= 20'h0003ff; // 64 words == 512 B
        3 : rd_addr_mask <= 20'h0007ff; // 128 words == 1 KB
        4 : rd_addr_mask <= 20'h000fff; // 256 words == 2 KB
        5 : rd_addr_mask <= 20'h001fff; // 512 words == 4 KB
        6 : rd_addr_mask <= 20'h003fff; // 1 Kwords == 8 KB
        7 : rd_addr_mask <= 20'h007fff; // 2 Kwords == 16 KB
        8 : rd_addr_mask <= 20'h00ffff; // 4 Kwords == 32 KB
        9 : rd_addr_mask <= 20'h01fff; // 8 Kwords == 64 KB
        10 : rd_addr_mask <= 20'h03fff; // 16 Kwords == 128 KB
        11 : rd_addr_mask <= 20'h07fff; // 32 Kwords == 256 KB
        12 : rd_addr_mask <= 20'h0ffff; // 64 Kwords == 512 KB
        13 : rd_addr_mask <= 20'h1fff; // 128 Kwords == 1024 KB
        14 : rd_addr_mask <= 20'h3fff; // 256 Kwords == 2048 KB
        15 : rd_addr_mask <= 20'h7fff; // 512 Kwords == 4096 KB
    endcase // case (rd_buf_size)
end // always @ (posedge clk)

//////////
// wr_buf_size_P1
//
always @(posedge clk) begin
    wr_buf_size_P1 <= (SY_BA_wr_buf_size);
end

//////////
// wr_addr_mask, wr_oc_ov_thresh
//
always @(posedge clk) begin
    case (wr_buf_size_P1)
        0 : wr_addr_mask <= 20'h0000ff; // 16 words == 128 B
        1 : wr_addr_mask <= 20'h0001ff; // 32 words == 256 B
        2 : wr_addr_mask <= 20'h0003ff; // 64 words == 512 B
        3 : wr_addr_mask <= 20'h0007ff; // 128 words == 1 KB
        4 : wr_addr_mask <= 20'h000fff; // 256 words == 2 KB
        5 : wr_addr_mask <= 20'h001fff; // 512 words == 4 KB
        6 : wr_addr_mask <= 20'h003fff; // 1 Kwords == 8 KB
        7 : wr_addr_mask <= 20'h007fff; // 2 Kwords == 16 KB
        8 : wr_addr_mask <= 20'h00ffff; // 4 Kwords == 32 KB
        9 : wr_addr_mask <= 20'h01fff; // 8 Kwords == 64 KB
        10 : wr_addr_mask <= 20'h03fff; // 16 Kwords == 128 KB
        11 : wr_addr_mask <= 20'h07fff; // 32 Kwords == 256 KB
        12 : wr_addr_mask <= 20'h0ffff; // 64 Kwords == 512 KB
        13 : wr_addr_mask <= 20'h1fff; // 128 Kwords == 1024 KB
        14 : wr_addr_mask <= 20'h3fff; // 256 Kwords == 2048 KB
        15 : wr_addr_mask <= 20'h7fff; // 512 Kwords == 4096 KB
    endcase
    wr_oc_ov_thresh <= (wr_addr_mask - 4); // 4 is approximate
end // always @ (posedge clk)

//////////
// wr_oc_full_thresh
//

```



```

always @(posedge clk) begin
  case (wr_buf_size_p1)
    0 : wr_oc_full_thresh <= 20'h00003; // 16 words == 128 B
    1 : wr_oc_full_thresh <= 20'h00013; // 32 words == 256 B
    2 : wr_oc_full_thresh <= 20'h0002f; // 64 words == 512 B
    3 : wr_oc_full_thresh <= 20'h0005f; // 128 words == 1 KB
    4 : wr_oc_full_thresh <= 20'h000bf; // 256 words == 2 KB
    5 : wr_oc_full_thresh <= 20'h0017f; // 512 words == 4 KB
    6 : wr_oc_full_thresh <= 20'h002ff; // 1 Kwords == 8 KB
    7 : wr_oc_full_thresh <= 20'h005ff; // 2 Kwords == 16 KB
    8 : wr_oc_full_thresh <= 20'h00bfff; // 4 Kwords == 32 KB
    9 : wr_oc_full_thresh <= 20'h017ff; // 8 Kwords == 64 KB
    10 : wr_oc_full_thresh <= 20'h02fff; // 16 Kwords == 128 KB
    11 : wr_oc_full_thresh <= 20'h05fff; // 32 Kwords == 256 KB
    12 : wr_oc_full_thresh <= 20'h0bfff; // 64 Kwords == 512 KB
    13 : wr_oc_full_thresh <= 20'h17fff; // 128 Kwords == 1024 KB
    14 : wr_oc_full_thresh <= 20'h2ffff; // 256 Kwords == 2048 KB
    15 : wr_oc_full_thresh <= 20'h5ffff; // 512 Kwords == 4096 KB
  endcase
end // always @ (posedge clk)

////////////////////
// wr_fifo_word
//
always @(posedge clk) begin
  case ({wr_fifo_word_hold_sel, wr_fifo_word_load_sel})
    0 : wr_fifo_word <= ({CF2_CMU_prty, CF2_CMU_data});
    1 : wr_fifo_word <= (wr_fifo_word_F1);
    2, 3 : wr_fifo_word <= (wr_fifo_word);
  endcase // case({wr_fifo_word_hold_sel, wr_fifo_word_load_sel})
end

////////////////////
// wr_fifo_read_P2, wr_fifo_SOCC, wr_fifo_EOCC
//
always @(posedge clk) begin
  case (reset_b)
    0 : begin
      wr_fifo_read_P2 <= 0;
      wr_fifo_SOCC <= 0;
      wr_fifo_EOCC <= 0;
    end
    1 : begin
      case ({wr_fifo_word_hold_sel, wr_fifo_word_load_sel})
        // FIFO underrun
        0 : begin
          wr_fifo_read_P2 <= (wr_fifo_read_P1);
          wr_fifo_SOCC <= (CF2_CMU_SOCC);
          wr_fifo_EOCC <= (CF2_CMU_EOCC);
        end
        // FIFO overrun
        1 : begin
          wr_fifo_read_P2 <= (wr_fifo_read_P2_F1);
          wr_fifo_SOCC <= (wr_fifo_SOCC_F1);
          wr_fifo_EOCC <= (wr_fifo_EOCC_F1);
        end
        // FIFO normal
        2, 3 : begin
          wr_fifo_read_P2 <= (wr_fifo_read_P2);
          wr_fifo_SOCC <= (wr_fifo_SOCC);
        end
      endcase
    end
  end
end

```

```

      wr_fifo_EOCC <= (wr_fifo_EOCC);
    end
    endcase // case({wr_fifo_word_hold_sel, wr_fifo_word_load_sel})
  end
  endcase // case(reset_b)
end // always @ (posedge clk)

////////////////////
// wr_fifo_word_F1
//
always @(posedge clk) begin
  case ({wr_fifo_word_F1_hold_sel,
        wr_fifo_word_F1_load_sel})
    0 : wr_fifo_word_F1 <= ({CF2_CMU_prty, CF2_CMU_data});
    1 : wr_fifo_word_F1 <= (wr_fifo_word_F1);
  endcase // case({wr_fifo_word_F1_hold_sel,
                  wr_fifo_word_F1_load_sel})
end

////////////////////
// wr_fifo_read_P2_F1, wr_fifo_SOCC_F1, wr_fifo_EOCC_F1
//
always @(posedge clk) begin
  case (reset_b)
    0 : begin
      wr_fifo_read_P2_F1 <= 0;
      wr_fifo_SOCC_F1 <= 0;
      wr_fifo_EOCC_F1 <= 0;
    end
    1 : begin
      case ({wr_fifo_word_F1_hold_sel,
            wr_fifo_read_P2_F1_load_sel})
        0 : begin
          wr_fifo_read_P2_F1 <= (wr_fifo_read_P1);
          wr_fifo_SOCC_F1 <= (wr_fifo_SOCC_F1);
          wr_fifo_EOCC_F1 <= (wr_fifo_EOCC_F1);
        end
        1 : begin
          wr_fifo_read_P2_F1 <= (wr_fifo_read_P2_F1);
          wr_fifo_SOCC_F1 <= (wr_fifo_SOCC_F1);
          wr_fifo_EOCC_F1 <= (wr_fifo_EOCC_F1);
        end
      endcase // case({wr_fifo_word_F1_hold_sel,
                    wr_fifo_read_P2_F1_load_sel})
    end
    end // case: 1
  endcase // case(reset_b)
end // always @ (posedge clk)

////////////////////
// Stage 1 Master
//
////////////////////
// CF2_CMU_oc_P1, CF3_CMU_oc_P1
//
always @(posedge clk) begin

```

```

CF2_CMU_oc_P1 <= CF2_CMU_oc;
CF3_CMU_oc_P1 <= CF3_CMU_oc;
end

////////////////////
// frozen
//

always @(posedge clk) begin: frozen_block
  case (reset_b)
    0 : frozen_dc <= (9'hfff);
    1 : begin
        case (frozen_dc.reset)
          0 : frozen_dc <= (frozen_dc - 1);
          1 : frozen_dc <= (9'hfff);
        endcase // case(frozen_reset_sel)
      end
    endcase // case(reset_b)
  end

  CMU_CIU_frozen <= (frozen_dc == 0);
end // always @ (posedge clk)

////////////////////
// bus_util_sel, bus_util, BA_CM_addr0, BA_CM_addr1
//

always @(re or
  we or
  addr_sel
) begin
  if ('re && !we) begin
    addr_sel_actual = 7;
    bus_util_sel = 0;
  end else begin
    addr_sel_actual = (addr_sel);
    if ((addr_sel == 4) || (addr_sel == 5)) begin
      bus_util_sel = 1;
    end else begin
      bus_util_sel = 0;
    end
  end
end

always @(posedge clk) begin
  case (reset_b)
    0 : bus_util <= 0;
    1 : begin
        case ({bus_util_counter_eq0, bus_util_sel})
          0 : bus_util <= (bus_util);
          1 : bus_util <= (bus_util + 1);
          2, 3 : bus_util <= 0;
        endcase // case({bus_util_counter_eq0, bus_util_sel})
      end
    endcase
  end // always @ (posedge clk)

  always @(posedge clk) begin
    BA_CM_addr0 <= BA_CM_addr_nxt;
    BA_CM_addr1 <= BA_CM_addr_nxt;
  end
end

```

```

always @(addr_sel_actual or
  CIU_CMU_dual_ppus or
  ptr_port_nxt or
  rd_hd_ptr_INDEX_rd_port_nxt or
  wr_tl_ptr_INDEX_wr_port_nxt or
  CIU_CMU_mem_addr
) begin

  case (addr_sel_actual)
    0 : begin
        case (CIU_CMU_dual_ppus && ptr_port_nxt[0])
          0 : BA_CM_addr_nxt = 'RD_HD_WR_TL_C1_PPU0_ADDR;
          1 : BA_CM_addr_nxt = 'RD_HD_WR_TL_C1_PPU1_ADDR;
        endcase // case(CIU_CMU_dual_ppus && ptr_port_nxt[0])
      end
    1 : begin
        case (CIU_CMU_dual_ppus && ptr_port_nxt[0])
          0 : BA_CM_addr_nxt = 'RD_HD_WR_TL_C2_PPU0_ADDR;
          1 : BA_CM_addr_nxt = 'RD_HD_WR_TL_C2_PPU1_ADDR;
        endcase // case(CIU_CMU_dual_ppus && ptr_port_nxt[0])
      end
    2 : begin
        case (CIU_CMU_dual_ppus && ptr_port_nxt[0])
          0 : BA_CM_addr_nxt = 'WR_HD_RD_TL_C1_PPU0_ADDR;
          1 : BA_CM_addr_nxt = 'WR_HD_RD_TL_C1_PPU1_ADDR;
        endcase // case(CIU_CMU_dual_ppus && ptr_port_nxt[0])
      end
    3 : begin
        case (CIU_CMU_dual_ppus && ptr_port_nxt[0])
          0 : BA_CM_addr_nxt = 'WR_HD_RD_TL_C2_PPU0_ADDR;
          1 : BA_CM_addr_nxt = 'WR_HD_RD_TL_C2_PPU1_ADDR;
        endcase // case(CIU_CMU_dual_ppus && ptr_port_nxt[0])
      end
    4 : BA_CM_addr_nxt = (rd_hd_ptr_INDEX_rd_port_nxt);
    5 : BA_CM_addr_nxt = (wr_tl_ptr_INDEX_wr_port_nxt);
    6 : BA_CM_addr_nxt = (CIU_CMU_mem_addr);
    7 : BA_CM_addr_nxt = 'SCRATCH_ADDR;
  endcase // case(addr_sel_actual)
end // always @ (posedge clk)

////////////////////
// bus_util_counter, bus_util_counter_eq0
//

always @(posedge clk) begin: bus_util_block

  case (reset_b)
    0 : bus_util_counter_nxt = 16'hffff;
    1 : bus_util_counter_nxt = (bus_util_counter - 1);
  endcase // case(reset_b)

  bus_util_counter <= bus_util_counter_nxt;

  bus_util_counter_eq0 <= (bus_util_counter_nxt == 0);
end

////////////////////
// CMU_CIU_bus_util
//

always @(posedge clk) begin
  case (reset_b)

```

```

0 : CMU_CIU_bus_util <= 0;
1 : begin
    case (bus_util_counter_eq0)
    0 : CMU_CIU_bus_util <= (CMU_CIU_bus_util);
    1 : CMU_CIU_bus_util <= (bus_util);
    endcase
end
endcase // case((reset_b, bus_util_counter_eq0))
end // always @ (posedge clk)

////////////////////
// port
//

always @(posedge clk) begin
    case (reset_b)
    0 : port <= 0;
    1 : port <= (port_nxt);
    endcase // case(reset_b)
end

////////////////////
// BA_CM_CE_b[0:3]
//

always @(posedge clk) begin: CE_b_block
    CE_b_nxt['NUM_PORTS-1:0] = ~(0);
    CE_tmp = (re || we || !CIU_CMU_use_mem_ce);

    case (reset_b)
    0 : ;
    1 : begin
        case (CIU_CMU_dual_ppus)
        0 : begin
            case (port_nxt)
            0 : CE_b_nxt[0] = !(CE_tmp);
            1 : CE_b_nxt[1] = !(CE_tmp);
            2 : CE_b_nxt[2] = !(CE_tmp);
            3 : CE_b_nxt[3] = !(CE_tmp);
            endcase
        end
        1 : begin
            case (port_nxt)
            0 : CE_b_nxt[0] = !(CE_tmp);
            1 : CE_b_nxt[1] = !(CE_tmp);
            2 : CE_b_nxt[2] = !(CE_tmp);
            3 : CE_b_nxt[3] = !(CE_tmp);
            endcase
        end
    end
    case (reset_b)
    0 : CE_b_nxt['NUM_PORTS-1:0] = ~(0);
    1 : begin
        case (CIU_CMU_use_mem_oe)
        0 : OE_b_nxt['NUM_PORTS-1:0] = ~(0);
        1 : begin
            case (CIU_CMU_dual_ppus)
            0 : begin
                OE_b_nxt[0] = !(re_port_rd_lat[0]);
                OE_b_nxt[1] = !(re_port_rd_lat[1]);
                OE_b_nxt[2] = !(re_port_rd_lat[2]);
                OE_b_nxt[3] = !(re_port_rd_lat[3]);
            end
            1 : begin
                OE_b_nxt[0] = !(re_port_rd_lat[0] ||
                    re_port_rd_lat[1]);
                OE_b_nxt[1] = !(re_port_rd_lat[2] ||
                    re_port_rd_lat[3]);
                OE_b_nxt[2] = 1;
                OE_b_nxt[3] = 1;
            end
        end
    endcase // case (CIU_CMU_dual_ppus)

```

```

WE_b_nxt['NUM_PORTS-1:0] = ~(0);
WE_tmp = (we || !(re_rd_lat_spread_prev && !CIU_CMU_use_mem_ce));

case (reset_b)
0 : ;
1 : begin
    case (CIU_CMU_dual_ppus)
    0 : begin
        case (port_nxt)
        0 : WE_b_nxt[0] = !(WE_tmp);
        1 : WE_b_nxt[1] = !(WE_tmp);
        2 : WE_b_nxt[2] = !(WE_tmp);
        3 : WE_b_nxt[3] = !(WE_tmp);
        endcase
    end
    1 : begin
        case (port_nxt)
        0 : WE_b_nxt[0] = !(WE_tmp);
        1 : WE_b_nxt[1] = !(WE_tmp);
        2 : WE_b_nxt[2] = !(WE_tmp);
        3 : WE_b_nxt[3] = !(WE_tmp);
        endcase
    end
    case (reset_b)
    0 : ;
    1 : begin
        case (CIU_CMU_use_mem_oe or
            re_port_rd_lat or
            BA_CM_OE_b_tmp
        ) begin: OE_b_block_ctrl
            case (reset_b)
            0 : OE_b_nxt['NUM_PORTS-1:0] = ~(0);
            1 : begin
                case (CIU_CMU_use_mem_oe)
                0 : OE_b_nxt['NUM_PORTS-1:0] = ~(0);
                1 : begin
                    case (CIU_CMU_dual_ppus)
                    0 : begin
                        OE_b_nxt[0] = !(re_port_rd_lat[0]);
                        OE_b_nxt[1] = !(re_port_rd_lat[1]);
                        OE_b_nxt[2] = !(re_port_rd_lat[2]);
                        OE_b_nxt[3] = !(re_port_rd_lat[3]);
                    end
                    1 : begin
                        OE_b_nxt[0] = !(re_port_rd_lat[0] ||
                            re_port_rd_lat[1]);
                        OE_b_nxt[1] = !(re_port_rd_lat[2] ||
                            re_port_rd_lat[3]);
                        OE_b_nxt[2] = 1;
                        OE_b_nxt[3] = 1;
                    end
                end
            endcase // case (CIU_CMU_dual_ppus)

```

```

end // case: 1
endcase // case(ciu_cmu_use_mem_oe)
end // case: 1
endcase // case(reset_b)
BA_CM_OE_b['NUM_PORTS-1:0] = (BA_CM_OE_b_tmp['NUM_PORTS-1:0] &
OE_b_nxt['NUM_PORTS-1:0]);
end // block: OE_b_block_ctrl

always @(posedge clk) begin: OE_b_block_seq
BA_CM_OE_b_tmp['NUM_PORTS-1:0] <= OE_b_nxt['NUM_PORTS-1:0];
end // always @ (posedge clk)

////////////////////////////////////
// wr_cyc_per_slot, rd_cyc_per_slot
//

always @(posedge clk) begin
rd_fifo_bias <= ({2'b0, CF2_CMU_oc_P1} +
{2'b0, CF3_CMU_oc_P1} -
('FIFO_SIZE - 2));
end

always @(CIU_CMU_cyc_per_entry or
CIU_CMU_cyc_per_slot or
CIU_CMU_wr_cyc_per_slot_min or
CIU_CMU_wr_cyc_per_slot_max or
rd_fifo_bias
) begin: fifo_bias_ctrl

case (CIU_CMU_cyc_per_entry)
// wr_cyc_per_slot = cnc_per_slot/2
0 : wr_cyc_per_slot_nxt_init =
({('FIFO_CALC-7(1'b0)), CIU_CMU_cyc_per_slot[7:1]});

// wr_cyc_per_slot = cnc_per_slot/2 + rd_fifo_bias/4
1 : wr_cyc_per_slot_nxt_init =
({(1{rd_fifo_bias['FIFO_CALC-1]}), // sign extension
rd_fifo_bias['FIFO_CALC-1:1]} +
({('FIFO_CALC-7(1'b0)), CIU_CMU_cyc_per_slot[7:1]});

// wr_cyc_per_slot = cnc_per_slot/2 + rd_fifo_bias/4
2 : wr_cyc_per_slot_nxt_init =
({(2{rd_fifo_bias['FIFO_CALC-1]}),
rd_fifo_bias['FIFO_CALC-1:2]} +
({('FIFO_CALC-7(1'b0)), CIU_CMU_cyc_per_slot[7:1]});

// wr_cyc_per_slot = cnc_per_slot/2 + rd_fifo_bias/8
3 : wr_cyc_per_slot_nxt_init =
({(3{rd_fifo_bias['FIFO_CALC-1]}),
rd_fifo_bias['FIFO_CALC-1:3]} +
({('FIFO_CALC-7(1'b0)), CIU_CMU_cyc_per_slot[7:1]});
endcase // case(CIU_CMU_cyc_per_entry)

wr_cyc_per_slot_underfl = ((wr_cyc_per_slot_nxt_init <
CIU_CMU_wr_cyc_per_slot_min) ||
wr_cyc_per_slot_nxt_init['FIFO_CALC-1]);

wr_cyc_per_slot_overfl = ((wr_cyc_per_slot_nxt_init >
CIU_CMU_wr_cyc_per_slot_max) &&
!wr_cyc_per_slot_nxt_init['FIFO_CALC-1]);

case (wr_cyc_per_slot_underfl, wr_cyc_per_slot_overfl)
0 : wr_cyc_per_slot_nxt = (wr_cyc_per_slot_nxt_init);
1 : wr_cyc_per_slot_nxt = (CIU_CMU_wr_cyc_per_slot_max);
2, 3 : wr_cyc_per_slot_nxt = (CIU_CMU_wr_cyc_per_slot_min);
endcase

end // block: fifo_bias_ctrl

always @(posedge clk) begin
case (wr_cyc_per_slot_sel) // (hold)
0 : wr_cyc_per_slot <= (wr_cyc_per_slot_nxt);
1 : wr_cyc_per_slot <= (wr_cyc_per_slot);
endcase

end

always @(posedge clk) begin
case (rd_cyc_per_slot_sel) // (hold)
0 : rd_cyc_per_slot <= (CIU_CMU_cyc_per_slot -
wr_cyc_per_slot_nxt);
1 : rd_cyc_per_slot <= (rd_cyc_per_slot);
endcase

end

////////////////////////////////////
// slot_dc, slot_dc_lteq0
//

always @(posedge clk) begin: slot_dc_block

case (slot_dc_sel)
0, 1 : slot_dc_nxt = (slot_dc - 1);
2 : slot_dc_nxt = (rd_cyc_per_slot - 1);
3 : slot_dc_nxt = (wr_cyc_per_slot - 1);
endcase // case(slot_dc_sel)

slot_dc <= (slot_dc_nxt);

slot_dc_lteq0 <= (slot_dc_nxt['FIFO_CALC-1] ||
(slot_dc_nxt == 0));

end

////////////////////////////////////
// ptr_port
//

always @(posedge clk) begin
case (reset_b)
0 : ptr_port <= 0;
1 : ptr_port <= (ptr_port_nxt);
endcase // case(reset_b)

end

////////////////////////////////////
// ptr_port_en
//

always @(posedge clk) begin
case (reset_b)
0 : ptr_port_en <= 0;
1 : ptr_port_en <= (ptr_port_en_nxt);
end

```

[illegible]

```

endcase // case(reset_b)
end

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
//
//
// Stage 2+CIU_CMU_rd_latency
//
//
//
//
//
//
//
//
//
//
// rd_word_F1
//
//
always @(posedge clk) begin
    rd_word_F1 <= ((CM_BA_prty, CM_BA_data));
end

```

```

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
//
//
// Stage 3+CIU_CMU_rd_latency
//
//
//
//
//
//
//
//
//
//
// rd_word
//
//

```

```

always @(posedge clk) begin
    rd_word_FAN_TOP <= (rd_word_F1);
    rd_word_FAN_0A <= (rd_word_F1);
    rd_word_FAN_0B <= (rd_word_F1);
    rd_word_FAN_0C <= (rd_word_F1);
    rd_word_FAN_1A <= (rd_word_F1);
    rd_word_FAN_1B <= (rd_word_F1);
    rd_word_FAN_1C <= (rd_word_F1);
    rd_word_FAN_2A <= (rd_word_F1);
    rd_word_FAN_2B <= (rd_word_F1);
    rd_word_FAN_2C <= (rd_word_F1);
    rd_word_FAN_3A <= (rd_word_F1);
    rd_word_FAN_3B <= (rd_word_F1);
    rd_word_FAN_3C <= (rd_word_F1);
end

```

```

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
//
//
// Stage 4+CIU_CMU_rd_latency
//
//
//
//
//
//
//
//
//
//

```

```

////////////////////////////////////
// CMU_CF3_data, CMU_CF3_prty (rd_word_F1)
//
//
always @(posedge clk) begin
    rd_word_F1 <= (rd_word_FAN_TOP);
    CMU_CF3_data <= (rd_word_FAN_TOP[63:0]);
    // Preserve parity error for discard by PM
    CMU_CF3_prty <= (!('rd_word_FAN_TOP[71:70]),
        !('rd_word_FAN_TOP[69:68]),
        !('rd_word_FAN_TOP[67:66]),
        !('rd_word_FAN_TOP[65:64]));
end

////////////////////////////////////
// rd_cell_len_dc, rd_cell_len_dc_eq0
//
always @(posedge clk) begin: rd_cell_len_block

    case (reset_b)
        0 : rd_cell_len_dc_nxt = 0;
        1 : begin
            case (rd_cell_len_dc_sel) // {hold, load}
                0 : rd_cell_len_dc_nxt = (rd_cell_len_dc - 1);
                1 : rd_cell_len_dc_nxt =
                    (rd_word_FAN_TOP['SCHED_CELL_LEN'] - 2);
                2, 3 : rd_cell_len_dc_nxt = (rd_cell_len_dc);
            endcase // case(rd_cell_len_dc_sel)
        end
    endcase // case(reset_b)

    rd_cell_len_dc <= rd_cell_len_dc_nxt;

    rd_cell_len_dc_eq0 <= (rd_cell_len_dc_nxt == 0);

end

////////////////////////////////////
// mem_data_out
//
always @(posedge clk) begin
    case (mem_rd_data_out_hold_sel)
        0 : CMU_CIU_mem_data_out <= (rd_word_FAN_TOP);
        1 : CMU_CIU_mem_data_out <= (CMU_CIU_mem_data_out);
    endcase
end

ba_cmu_port_dp ba_cmu_port_dp_inst_0
// Outputs
    .wr_oc_full_MEM (wr_oc_full[0]),
    .wr_oc_ov_MEM (wr_oc_ov[0]),
    .rd_word_wr_hd_ptr_eq_wr_hd_ptr_MEM (rd_word_wr_hd_ptr_eq_wr_hd_ptr[0]),
    .port_st_MEM (port_st_0),
    .reload_ack_MEM (CMU_CIU_reload_ack[0]),
    .rd_port_locked_MEM (rd_port_locked[0]),
    .rd_oc_eq0_MEM (rd_oc_eq0[0]),

```

```

.wr_ptr_locked_MEM (wr_ptr_locked[0]),
.timeout_next_MEM (CMU_CIU_eq0[1]),
.rd_hd_ptr_MEM (rd_hd_ptr[0]),
.rd_hd_ptr_shadow_MEM (rd_hd_ptr_shadow_0),
.wr_oc_MEM (CMU_CIU_wr_oc_0),
.wr_tl_ptr_MEM (wr_tl_ptr[0]),
.wr_tl_ptr_shadow_MEM (wr_tl_ptr_shadow_0),
// Inputs
.clk (clk),
.reset_b (reset_b),
.wr_addr_mask (wr_addr_mask),
.wr_oc_full_thresh (wr_oc_full_thresh),
.rd_word_FAN_A (rd_word_FAN_0A),
.rd_word_FAN_B (rd_word_FAN_0B),
.rd_word_FAN_C (rd_word_FAN_0C),
.ppu_timeout (CIU_CMU_ppu_timeout),
.rd_hd_ptr_rd_lat (rd_hd_ptr_rd_lat),
.rd_addr_mask (rd_addr_mask),
.rd_oc_rd_lat (rd_oc_rd_lat),
.ptr_port_updated_MEM (ptr_port_updated[0]),
.ppu_ic_reset_ptr_MEM (ppu_ic_reset_ptr[0]),
.rd_ptr_locked_next_MEM (rd_ptr_locked_next[0]),
.rd_ptr_load_sel_MEM (rd_ptr_load_sel[0]),
.rd_hd_ptr_roll_sel_MEM (rd_hd_ptr_roll_sel[0]),
.rd_hd_ptr_hold_sel_MEM (rd_hd_ptr_hold_sel[0]),
.rd_oc_roll_sel_MEM (rd_oc_roll_sel[0]),
.rd_oc_load_sel_MEM (rd_oc_load_sel[0]),
.rd_oc_hold_sel_MEM (rd_oc_hold_sel[0]),
.wr_ptrs_reloaded_set_sel_MEM (wr_ptrs_reloaded_set_sel[0]),
.wr_ptr_locked_next_MEM (wr_ptr_locked_next[0]),
.wr_ptrs_reloaded_sel_MEM (wr_ptrs_reloaded_sel[0]),
.wr_tl_ptr_load_sel_MEM (wr_tl_ptr_load_sel[0]),
.wr_tl_ptr_hold_sel_MEM (wr_tl_ptr_hold_sel[0]),
.wr_oc_hold_sel_MEM (wr_oc_hold_sel[0]),
.wr_oc_load_sel_MEM (wr_oc_load_sel[0]),
.wr_tl_ptr_shadow_load_sel_MEM (wr_tl_ptr_shadow_load_sel[0]),
.wr_tl_ptr_shadow_hold_sel_MEM (wr_tl_ptr_shadow_hold_sel[0]),
.reload_req_MEM (CIU_CMU_reload_req[0]),
.corrupted_ptr_next_MEM (CMU_CIU_corrupted_ptr_next[0]),
.sync_lost_next_MEM (CMU_CIU_sync_lost_next[0]),
.mem_access (CIU_CMU_mem_access),
.rd_hd_ptr_shadow_load_sel_MEM (rd_hd_ptr_shadow_load_sel[0]),
.rd_hd_ptr_shadow_hold_sel_MEM (rd_hd_ptr_shadow_hold_sel[0]),
);

ba_cmu_port_dp ba_cmu_port_dp_inst_1
// Outputs
.wr_oc_full_MEM (wr_oc_full[1]),
.wr_oc_ov_MEM (wr_oc_ov[1]),
.rd_word_wr_hd_ptr_eq_wr_hd_ptr_MEM (rd_word_wr_hd_ptr_eq_wr_hd_ptr[1]),
.port_st_MEM (port_st_1),
.reload_ack_MEM (CMU_CIU_reload_ack[1]),
.rd_ptr_locked_MEM (rd_ptr_locked[1]),
.rd_oc_eq0_MEM (rd_oc_eq0[1]),
.wr_ptr_locked_MEM (wr_ptr_locked[1]),
.wr_oc_eq0_MEM (wr_oc_eq0[1]),
.timeout_next_MEM (CMU_CIU_timeout_next[1]),
.rd_hd_ptr_MEM (rd_hd_ptr_1),
.wr_oc_MEM (CMU_CIU_wr_oc_2),
.wr_tl_ptr_MEM (wr_tl_ptr_2),
.wr_tl_ptr_shadow_MEM (wr_tl_ptr_shadow_1),
.rd_oc_MEM (CMU_CIU_rd_oc_1),

```

```

.wr_oc_MEM (CMU_CIU_wr_oc_1),
.wr_tl_ptr_MEM (wr_tl_ptr_1),
.wr_tl_ptr_shadow_MEM (wr_tl_ptr_shadow_1),
// Inputs
.clk (clk),
.reset_b (reset_b),
.wr_addr_mask (wr_addr_mask),
.wr_oc_full_thresh (wr_oc_full_thresh),
.wr_oc_ov_thresh (wr_oc_ov_thresh),
.rd_word_FAN_A (rd_word_FAN_1A),
.rd_word_FAN_B (rd_word_FAN_1B),
.rd_word_FAN_C (rd_word_FAN_1C),
.ppu_timeout (CIU_CMU_ppu_timeout),
.rd_hd_ptr_rd_lat (rd_hd_ptr_rd_lat),
.rd_addr_mask (rd_addr_mask),
.rd_oc_rd_lat (rd_oc_rd_lat),
.ptr_port_updated_MEM (ptr_port_updated[1]),
.ppu_ic_reset_ptr_MEM (ppu_ic_reset_ptr[1]),
.rd_ptr_locked_next_MEM (rd_ptr_locked_next[1]),
.rd_ptr_load_sel_MEM (rd_ptr_load_sel[1]),
.rd_hd_ptr_roll_sel_MEM (rd_hd_ptr_roll_sel[1]),
.rd_hd_ptr_hold_sel_MEM (rd_hd_ptr_hold_sel[1]),
.rd_oc_roll_sel_MEM (rd_oc_roll_sel[1]),
.rd_oc_load_sel_MEM (rd_oc_load_sel[1]),
.rd_oc_hold_sel_MEM (rd_oc_hold_sel[1]),
.wr_ptrs_reloaded_set_sel_MEM (wr_ptrs_reloaded_set_sel[1]),
.wr_ptr_locked_next_MEM (wr_ptr_locked_next[1]),
.wr_ptrs_reloaded_sel_MEM (wr_ptrs_reloaded_sel[1]),
.wr_tl_ptr_load_sel_MEM (wr_tl_ptr_load_sel[1]),
.wr_tl_ptr_hold_sel_MEM (wr_tl_ptr_hold_sel[1]),
.wr_oc_hold_sel_MEM (wr_oc_hold_sel[1]),
.wr_oc_load_sel_MEM (wr_oc_load_sel[1]),
.wr_tl_ptr_shadow_load_sel_MEM (wr_tl_ptr_shadow_load_sel[1]),
.wr_tl_ptr_shadow_hold_sel_MEM (wr_tl_ptr_shadow_hold_sel[1]),
.reload_req_MEM (CIU_CMU_reload_req[1]),
.corrupted_ptr_next_MEM (CMU_CIU_corrupted_ptr_next[1]),
.sync_lost_next_MEM (CMU_CIU_sync_lost_next[1]),
.mem_access (CIU_CMU_mem_access),
.rd_hd_ptr_shadow_load_sel_MEM (rd_hd_ptr_shadow_load_sel[1]),
.rd_hd_ptr_shadow_hold_sel_MEM (rd_hd_ptr_shadow_hold_sel[1]),
);

ba_cmu_port_dp ba_cmu_port_dp_inst_2
// Outputs
.wr_oc_full_MEM (wr_oc_full[2]),
.wr_oc_ov_MEM (wr_oc_ov[2]),
.rd_word_wr_hd_ptr_eq_wr_hd_ptr_MEM (rd_word_wr_hd_ptr_eq_wr_hd_ptr[2]),
.port_st_MEM (port_st_2),
.reload_ack_MEM (CMU_CIU_reload_ack[2]),
.rd_ptr_locked_MEM (rd_ptr_locked[2]),
.rd_oc_eq0_MEM (rd_oc_eq0[2]),
.wr_ptr_locked_MEM (wr_ptr_locked[2]),
.wr_oc_eq0_MEM (wr_oc_eq0[2]),
.timeout_next_MEM (CMU_CIU_timeout_next[2]),
.rd_hd_ptr_MEM (rd_hd_ptr_2),
.rd_hd_ptr_shadow_MEM (rd_hd_ptr_shadow_2),
.rd_oc_MEM (CMU_CIU_rd_oc_2),
.wr_oc_MEM (CMU_CIU_wr_oc_2),
.wr_tl_ptr_MEM (wr_tl_ptr_2),
.wr_tl_ptr_shadow_MEM (wr_tl_ptr_shadow_2),
.clk (clk),
.reset_b (reset_b),

```

07/12/00
16:59:14

```
.wr_addr_mask (wr_addr_mask),
.wr_oc_full_thresh (wr_oc_full_thresh),
.wr_oc_ov_thresh (wr_oc_ov_thresh),
.rd_word_FAN_A (rd_word_FAN_2A),
.rd_word_FAN_B (rd_word_FAN_2B),
.rd_word_FAN_C (rd_word_FAN_2C),
.ppu_timeout (CIU_CMU_ppu_timeout),
.rd_hd_ptr_rd_lat (rd_hd_ptr_rd_lat),
.rd_addr_mask (rd_addr_mask),
.rd_oc_rd_lat (rd_oc_rd_lat),
.ptr_port_updated_MEM (ptr_port_updated[2]),
.ppu_ic_reset_ptr_MEM (ppu_ic_reset_ptr[2]),
.rd_port_locked_nxt_MEM (rd_port_locked_nxt[2]),
.rd_hd_ptr_load_sel_MEM (rd_hd_ptr_load_sel[2]),
.rd_hd_ptr_rol_sel_MEM (rd_hd_ptr_rol_sel[2]),
.rd_hd_ptr_hold_sel_MEM (rd_hd_ptr_hold_sel[2]),
.rd_oc_rol_sel_MEM (rd_oc_rol_sel[2]),
.rd_oc_load_sel_MEM (rd_oc_load_sel[2]),
.rd_oc_hold_sel_MEM (rd_oc_hold_sel[2]),
.wr_ptrs_reloaded_set_sel_MEM (wr_ptrs_reloaded_set_sel[2]),
.wr_ptrs_reloaded_nxt_MEM (wr_ptrs_reloaded_nxt[2]),
.wr_hd_ptr_load_sel_MEM (wr_hd_ptr_load_sel[2]),
.wr_tl_ptr_load_sel_MEM (wr_tl_ptr_load_sel[2]),
.wr_tl_ptr_hold_sel_MEM (wr_tl_ptr_hold_sel[2]),
.wr_oc_hold_sel_MEM (wr_oc_hold_sel[2]),
.wr_oc_load_sel_MEM (wr_oc_load_sel[2]),
.wr_tl_ptr_shadow_load_sel_MEM (wr_tl_ptr_shadow_load_sel[2]),
.wr_tl_ptr_shadow_hold_sel_MEM (wr_tl_ptr_shadow_hold_sel[2]),
.reload_req_MEM (CIU_CMU_reload_req[2]),
.sync_lost_nxt_MEM (CMU_CIU_sync_lost_nxt[2]),
.mem_access (CIU_CMU_mem_access),
.rd_hd_ptr_shadow_load_sel_MEM (rd_hd_ptr_shadow_load_sel[2]),
.rd_hd_ptr_shadow_hold_sel_MEM (rd_hd_ptr_shadow_hold_sel[2])
);

ba_cmu_port_dp ba_cmu_port_dp_inst_3
// Outputs
.wr_oc_full_MEM (wr_oc_full[3]),
.wr_oc_ov_MEM (wr_oc_ov[3]),
.rd_word_wr_hd_ptr_eq_wr_hd_ptr_MEM (rd_word_wr_hd_ptr_eq_wr_hd_ptr[3]),
.port_st_MEM (port_st_3),
.reload_ack_MEM (CMU_CIU_reload_ack[3]),
.rd_port_locked_MEM (rd_port_locked[3]),
.rd_oc_eq0_MEM (rd_oc_eq0[3]),
.wr_port_locked_MEM (wr_port_locked[3]),
.wr_oc_eq0_MEM (wr_oc_eq0[3]),
.timeout_nxt_MEM (CMU_CIU_timeout_nxt[3]),
.rd_hd_ptr_MEM (rd_hd_ptr_3),
.rd_hd_ptr_shadow_MEM (rd_hd_ptr_shadow_3),
.rd_oc_MEM (CMU_CIU_rd_oc_3),
.wr_oc_MEM (CMU_CIU_wr_oc_3),
.wr_tl_ptr_MEM (wr_tl_ptr_3),
.wr_tl_ptr_shadow_MEM (wr_tl_ptr_shadow_3),
// Inputs
.clk (clk),
.reset_b (reset_b),
.wr_addr_mask (wr_addr_mask),
.wr_oc_full_thresh (wr_oc_full_thresh),
.rd_word_FAN_A (rd_word_FAN_3A),
.rd_word_FAN_B (rd_word_FAN_3B),
.rd_word_FAN_C (rd_word_FAN_3C),
```

```
.ppu_timeout (CIU_CMU_ppu_timeout),
.rd_hd_ptr_rd_lat (rd_hd_ptr_rd_lat),
.rd_addr_mask (rd_addr_mask),
.rd_oc_rd_lat (rd_oc_rd_lat),
.ppu_ic_reset_ptr_MEM (ppu_ic_reset_ptr[3]),
.rd_port_locked_nxt_MEM (rd_port_locked_nxt[3]),
.rd_hd_ptr_load_sel_MEM (rd_hd_ptr_load_sel[3]),
.rd_hd_ptr_rol_sel_MEM (rd_hd_ptr_rol_sel[3]),
.rd_hd_ptr_hold_sel_MEM (rd_hd_ptr_hold_sel[3]),
.rd_oc_rol_sel_MEM (rd_oc_rol_sel[3]),
.rd_oc_load_sel_MEM (rd_oc_load_sel[3]),
.rd_oc_hold_sel_MEM (rd_oc_hold_sel[3]),
.wr_ptrs_reloaded_set_sel_MEM (wr_ptrs_reloaded_set_sel[3]),
.wr_ptrs_reloaded_nxt_MEM (wr_ptrs_reloaded_nxt[3]),
.wr_hd_ptr_load_sel_MEM (wr_hd_ptr_load_sel[3]),
.wr_tl_ptr_load_sel_MEM (wr_tl_ptr_load_sel[3]),
.wr_tl_ptr_hold_sel_MEM (wr_tl_ptr_hold_sel[3]),
.wr_oc_hold_sel_MEM (wr_oc_hold_sel[3]),
.wr_oc_load_sel_MEM (wr_oc_load_sel[3]),
.wr_tl_ptr_shadow_load_sel_MEM (wr_tl_ptr_shadow_load_sel[3]),
.wr_tl_ptr_shadow_hold_sel_MEM (wr_tl_ptr_shadow_hold_sel[3]),
.reload_req_MEM (CIU_CMU_reload_req[3]),
.corrupted_ptr_nxt_MEM (CMU_CIU_corrupted_ptr_nxt[3]),
.sync_lost_nxt_MEM (CMU_CIU_sync_lost_nxt[3]),
.mem_access (CIU_CMU_mem_access),
.rd_hd_ptr_shadow_load_sel_MEM (rd_hd_ptr_shadow_load_sel[3]),
.rd_hd_ptr_shadow_hold_sel_MEM (rd_hd_ptr_shadow_hold_sel[3])
);

endmodule // ba_cmu_core
```

```
=====
// This Software and Related Documentation contain the trade secret
// and confidential information of Celox Communications, Inc.
// Copyright (c) 1999 Celox Communications, Inc., St. Louis, MO
// Unpublished.
// All Rights Reserved.
//
// Restricted Rights Legend: The Software and Documentation
// are provided with Restricted Rights. Use, Duplication, or Disclosure
// by the Government is Subject to Restrictions as Set Forth in
// Paragraph (c)(1)(ii) of the Rights in Technical Data and
// Computer Software Clause at DFARS 252.227-7013. Contractor/Manufacturer
// is Celox Communications, Inc., 4041 Forest Park Blvd., St. Louis, MO 63108.
```


03/30/00
02:43:07

ba_cmu_port_dp.N3

1

```
// =====
// File Name      : $RCSfile: ba_cmu_port_dp.v,v $
// Version        : $Revision: 1.31 $
// Status         : In beta, needs a few features, more testing
// Module         : Control Memory Unit Port Datapath (ba_cmu_port_dp)
//
// Author         : Andrew Ryan
// E-mail         : acryan@wipinfo.soft.net, acryan@celoxcom.com
// Phone         : USA, 1-314-615-6338
// Company        : Celox Communication Corporation
// Creation Date  : August 30, 1999
// Last Modified by : $Author: acryan $
// Last Modified on : $Date: 2000/03/30 08:43:07 $
//
// Dependencies   : none
//
// Description    : The Control Memory Unit (CMU) transfers control cells
//                  to and scheduled cells from the PPU's. It must make
//                  sure that the FIFOs on one side do not underflow/
//                  overflow, and that the bandwidth to the dual-ported
//                  SRAMs is used optimally so that one misbehaving PPU
//                  does not adversely affect the other PPU's connected to
//                  this unit.
//
// Simulator      : VCS and Verilog-XL
//
// Simulation Notes : Here's the a diagram of the test bench:
//
//   fifo_2  -->  dp_sram  <->  dum_ppu
//   cell_gen  <-->  ba_cmu  <-->  dum_ppu
//   fifo_3  <-->  dp_sram  <->  dum_ppu
//
// The cell_gen sends a variable traffic of packets
// thru fifo_2, thru the ba_cmu, thru the dp_ram's,
// to the dum_ppu's. The dum_ppu's then reformat
// and resend the traffic back to the cell_gen,
// which keeps track of cells, and flags cells that
// haven't returned after its in-flight-cell buffer
// overflows.
//
// It's ok if a port times out, but you should be
// able to keep it from happening by making the dum_ppu's
// more responsive (lower their pink noise dampening
// factor).
//
// Synthesis Notes : Needs more development...
//
// Application Notes : Oh my, not now...
//
// Limitations      : N/A
//
// Bugs, Open issues, future enhancements :
//
// make parameterized dp_sram.pro.v, adding rd_trash_cyc.
// test rd_to_rd_wait = 0.
// check read port scheduling fairness.
// make sure wait states are followed, i.e. no bus contention.
// test to see if unit recovers gracefully from exceptions.
//
// References       : Refer to the soon-to-be-completed docs in /Docs/...
//
// Disclaimer       : Copyright © 1999 Celox Communications Corporation
//                  All rights reserved
```

```
//
// Revision History : N/A
//
// $Log: ba_cmu_port_dp.v,v $
// Revision 1.31 2000/03/30 08:43:07 acryan
// Deleted code no-effect code accidentally cut-and-pasted in port status sched.
//
// Revision 1.30 2000/03/29 10:38:00 acryan
// Fixed bug where pointers would not update after a cell transaction if
// port was shut down.
// Removed ptr_port_locked* variables.
//
// Revision 1.29 2000/03/28 16:52:02 acryan
// Fixed read scheduler bug from last commit.
// Fixed port shutdown and boot up design flaw. This manifested itself because
// the CMU didn't do a final update of the pointers before shutting down a port,
// causing cells to be lost and other cells to transmit twice.
//
// Revision 1.28 2000/03/27 10:33:38 acryan
// Made a few more changes for code coverage reasons.
// Removed impossible cases from code in port_dp (I hope).
//
// Revision 1.27 2000/02/19 05:43:44 acryan
// Removed several critical paths by:
// 1. Flopping wr_buf_size.
// 2. Flopping rd_fifo_bias.
// 3. Flopping wr_oc_full, wr_oc_ov.
// 4. Tested with 1M cells.
//
// Revision 1.26 2000/02/10 09:09:54 acryan
// Reduced code paths.
// Reduced some (hopefully) redundant logic.
// Made dum_mc do indirect memory access after every reboot of BA.
//
// Revision 1.25 2000/02/09 15:44:38 acryan
// Fixed totally obscure bug that happens only when the read tail and
// read head pointers wraparound and the occupancy becomes zero and
// ready latency is large and other conditions.
//
// Revision 1.24 2000/02/09 12:49:44 acryan
// The CMU now issues fifo_sync_lost interrupt in addition if:
// 1. An SOCC comes after another SOCC without a EOCC between.
// 2. If any data comes between an EOCC and SOCC.
// The write scheduler had to be re-written to support this.
// The CMU has been tested with 100K cells afterwards.
// The rd_new_port_dc is now 20 bits wide instead of 16 bits,
// to reduce the possibility of wrapping around on large cells.
// I renamed *_next to *_nxt.
// I renamed wr_scheduled to wr_sched_get_word.
// I added lots of comments in the code for clarity.
//
// Revision 1.23 2000/02/09 05:02:06 spatel
//
// Added Ambit directive for specifying synchronous reset signals.
//
// Revision 1.22 2000/02/08 10:49:31 acryan
// Made all testbench parameters into defines.
// Fixed a bug in write scheduler, wr_port should have been wr_port_next.
// Fixed testbench bug where under light loads, cells would have large latencies.
// Fixed outgoing parity so it reflects bad parity from DPRAMs.
// Modified cell_gen so cells can have a timeout, like 1 ms.
//
// Revision 1.21 2000/02/07 11:14:44 acryan
// Fixed bug in read gatherer that would miss port change signals,
// which might cause it not to issue interrupts when it should.
```

03/30/00
02:43:07

ba1cmu_port_dp.N

2

```
// Also reduced the number of code paths, for coverage observability.
// Revision 1.20 2000/02/05 12:30:53 acryan
// Changed ppu_dc to ppu_ic, making timeout param more responsive.
//
// Revision 1.19 2000/02/04 15:56:30 acryan
// Made rd_ptr_locked unlock much faster if port is CLOSED.
// Made port status scheduler more observable for code coverage.
//
// Revision 1.18 2000/01/25 06:17:34 celox
// Extended rd_word to be two cycles long, for timing closure purposes.
//
// Revision 1.17 2000/01/16 04:03:49 celox
// Changed corrupted pointers from 2-bit to 3-bit saturating counter, should
// reduce false positives from every 1/4 second to every 1 year.
// Made read and write buffer occupancies reset on reset_b, for appearance's sake.
// Made CMU capable of handling single-word cells in both directions.
//
// Revision 1.16 2000/01/09 02:51:13 celox
// Removed lots of timescales.
// Also made CMU more observable for code coverage purposes.
//
// Revision 1.15 2000/01/08 08:35:09 celox
// Fixed indirect memory access bug.
// Fixed write tail pointer bug.
//
// Revision 1.14 2000/01/08 03:15:28 celox
// Added rd_hd_ptr_shadow to fix bug.
// Fixed double pointer reload bug.
// Broke up seq blocks into smaller blocks for code coverage.
//
// Revision 1.13 1999/12/20 09:50:08 acryan
// Made test bench able to change config params on the fly and to
// reset the CMU occasionally. Fixed several reset-related bugs
// in the test bench and RTL.
//
// Revision 1.12 1999/12/17 16:25:20 acryan
// Implemented and tested indirect memory access.
//
// Revision 1.11 1999/12/17 04:23:06 acryan
// Fixed potential bug of bus driver not delayed by wr_latency.
//
// Revision 1.10 1999/12/16 10:41:07 acryan
// Made a few minor changes for synthesis and code coverage purposes.
//
// Revision 1.9 1999/12/15 15:20:20 acryan
// 1. Implemented and tested HALTED ports able to become ACTIVE by receiving
// all cells, making write buffer occupancy zero.
// 2. Implemented HALTED ports' cells being redirected.
// 3. Implemented B bit packet continuity.
// 4. Implemented fifo sync lost logic, and the write scheduler's ability
// to handle and discard extra large packets.
// 5. Cleaned up the write scheduler code, reduced state.
// 6. Implemented write data state to force the data bus to low impedance.
// 7. Main FSM skips State 0 because of #6.
// 8. Renamed rd_signature config param to signature.
//
// Revision 1.8 1999/12/09 10:07:25 acryan
// Changed slot_dc_eq_0 to slot_dc_lteq_0, which should fix lots of scheduling
// glitches.
//
// Revision 1.7 1999/12/08 16:40:17 acryan
// Renamed reload to reload_req.
//
// Revision 1.6 1999/12/08 15:42 44 acryan

// Fixed parity error bug.
// Revision 1.5 1999/12/01 08:44:47 acryan
// Changed ptr_s to ptr_shadow for clarity.
//
// Revision 1.4 1999/11/30 06:23:22 acryan
// Rewrote the read scheduler.
// Tightened bus timing.
// Fixed rd_to_rd_wait_cyc=0 bug.
//
// Revision 1.3 1999/11/29 12:51:14 acryan
// Put anti-floating data bus measures into main FSM in CMU.
//
// Revision 1.2 1999/11/22 06:16:57 acryan
// Added some header comments.
//
// Revision 1.1 1999/11/19 09:21:37 acryan
// Rewrote CMU to make it more synthesizable. Also fixed ppu_timeout bug.
//
// =====
//
// include "defines_ba_cmu.vh"
//
// ambit synthesis set_reset synchronous signals = "reset_b"
//
module ba_cmu_port_dp (*AutoArg*)
// Outputs
wr_oc_full_MEM, wr_oc_ov_MEM, rd_word.wr_hd_ptr_eq.wr_hd_ptr_MEM,
port_st_MEM, reload_ack_MEM, rd_ptr_locked_MEM, rd_oc_eq0_MEM,
wr_ptr_locked_MEM, wr_oc_eq0_MEM, timeout_nxt_MEM, rd_hd_ptr_MEM,
rd_hd_ptr_shadow_MEM, rd_oc_MEM, wr_oc_MEM, wr_tl_ptr_MEM,
wr_tl_ptr_shadow_MEM,
// Inputs
clk, reset_b, wr_addr_mask, wr_oc_full_thresh, wr_oc_ov_thresh,
rd_word.FAN_A, rd_word.FAN_B, rd_addr_mask, rd_oc_rd_lat,
rd_hd_ptr_rd_lat, rd_ptr_updated_MEM, ppu_ic_reset_ptr_MEM,
ptr_ptr_updated_MEM, rd_ptr_locked_nxt_MEM, rd_hd_ptr_load_sel_MEM,
rd_ptr_rollback_sel_MEM, rd_hd_ptr_hold_sel_MEM,
rd_oc_rollback_sel_MEM, rd_oc_load_sel_MEM, rd_oc_hold_sel_MEM,
rd_ptrs_reloaded_sel_sel_MEM, wr_ptr_locked_nxt_MEM,
wr_ptrs_reloaded_sel_sel_MEM, wr_hd_ptr_load_sel_MEM,
wr_tl_ptr_load_sel_MEM, wr_tl_ptr_hold_sel_MEM,
wr_oc_hold_sel_MEM, wr_oc_load_sel_MEM,
wr_tl_ptr_shadow_load_sel_MEM, wr_tl_ptr_shadow_hold_sel_MEM,
reload_req_MEM, corrupted_ptr_nxt_MEM, sync_lost_nxt_MEM,
mem_access, rd_hd_ptr_shadow_load_sel_MEM,
rd_hd_ptr_shadow_hold_sel_MEM
);
input clk;
input reset_b;

input [19:0] wr_addr_mask;
input [19:0] wr_oc_full_thresh;
input [19:0] wr_oc_ov_thresh;
input [71:0] rd_word.FAN_A;
input [71:0] rd_word.FAN_B;
input [71:0] rd_word.FAN_C;
input [15:0] ppu_timeout;
input [19:0] rd_hd_ptr_rd_lat;
input [19:0] rd_addr_mask;
input [19:0] rd_oc_rd_lat;
```



ba.comport.dp.v

```

reg [19:0] wr_oc_mem;
reg [19:0] wr_tl_ptr_mem;
reg [19:0] wr_tl_ptr_shadow_mem;
reg [19:0] rd_hd_ptr_shadow_mem;

reg [31:0] ppu_ic_mem;
reg [19:0] wr_hd_ptr_mem;

reg [19:0] wr_tl_ptr_mem_nxt;
reg [19:0] wr_oc_nxt_mem;
reg [19:0] rd_oc_nxt_mem;

//////////////////////////
//////////////////////////
////
///// Port status scheduler
/////
//////////////////////////
//////////////////////////

always @(reload_req_mem or
        reload_ack_mem or
        port_st_mem or
        rd_port_locked_nxt_mem or
        wr_port_locked_nxt_mem or
        ptr_port_updated_mem or
        corrupted_ptr_nxt_mem or
        sync_lost_nxt_mem or
        rd_ptrs_reloaded_mem or
        wr_ptrs_reloaded_mem or
        ppu_ic_ov_mem or
        wr_oc_eq0_mem
        ) begin: port_st_ctrl

    reload_ack_nxt_mem = reload_ack_mem;
    rd_ptrs_reloaded_clear_sel_mem = 0;
    wr_ptrs_reloaded_clear_sel_mem = 0;
    port_st_nxt_mem = port_st_mem;
    timeout_nxt_mem = 0;

    case (port_st_mem)
        ////////////////////////////////////////////
        // If port is INACTIVE...
        //
        // The INACTIVE status comes right before the CLOSING status.
        // While a port is INACTIVE, it reading or writing its
        // last atomic cell/packet transaction.
        //
        'INACTIVE : begin
            // Once the INACTIVE port is not reading or writing data,
            // but is updating pointers, make CLOSED.
            if ('rd_port_locked_nxt_mem && 'wr_port_locked_nxt_mem &&
                ptr_port_updated_mem) begin
                port_st_nxt_mem = 'CLOSING;
            end
        end
    end
end

```

03/30/00
02:43:07

```
// If port is CLOSING...
//
// The CLOSING status comes right before the CLOSED status.
// While a port is CLOSING, the pointers will be read and
// written one more time.
//
'CLOSING : begin
// Make sure pointers are written one more time
if (ptr_port_updated_MEM) begin
    port_st_nxt_MEM = 'CLOSED;
end
end // case: 'CLOSING

////////////////////
// If port is CLOSED...
//
// CLOSED is the default status of each port.
// While a port is CLOSED, the pointers can be read and
// written only once.
//
'CLOSED : begin
// Once the CLOSED port's pointers written, send ack.
if (reload_req_MEM && 'reload_ack_MEM) begin
    reload_ack_nxt_MEM = 1;
end else begin
    // When PPU controller finished, it lowers reload,
    // port = RELOAD.
    if (!reload_req_MEM && reload_ack_MEM) begin
        port_st_nxt_MEM = 'RELOADING;
        rd_ptrs_reloaded_clear_sel_MEM = 1;
        wr_ptrs_reloaded_clear_sel_MEM = 1;
    end
end
end // case: 'CLOSED

////////////////////
// If port is RELOADING...
//
// The RELOAD status is when a port is attempted to load
// all four pointers from memory successfully.
//
'RELOADING : begin
// If PPU controller requests PPU reload, mark port CLOSED.
if (reload_req_MEM) begin
    port_st_nxt_MEM = 'CLOSED;
end else begin
    // If reloading pointers fails, port = CLOSED, lower ack.
    if (corrupted_ptr_nxt_MEM) begin
        reload_ack_nxt_MEM = 0;
        port_st_nxt_MEM = 'CLOSED;
    end else begin
        // When finished reloading pointers, port = ACTIVE,
        // lower ack.
        if (wr_ptrs_reloaded_MEM &&
            rd_ptrs_reloaded_MEM) begin
            reload_ack_nxt_MEM = 0;
            port_st_nxt_MEM = 'ACTIVE;
        end
    end
end
```

03/30/00 02:43:07

```
ba_cmu_dp.wr
// If port is ACTIVE...
//
// The ACTIVE status is normal operation of a port.
//
'ACTIVE : begin
// If read gatherer loses sync, mark port INACTIVE.
// If PPU controller requests PPU reload, mark port INACTIVE.
if (sync_lost_nxt_MEM || reload_req_MEM) begin
    port_st_nxt_MEM = 'INACTIVE;
end else begin
    // If PPU times out, set port to HALTED.
    if (ppu_ic_ov_MEM) begin
        port_st_nxt_MEM = 'HALTED;
        timeout_nxt_MEM = 1;
    end
end
end // case: 'ACTIVE

////////////////////
// If port is HALTED...
//
// A port becomes HALTED if the PPU shows no signs of life.
//
'HALTED : begin
// If read gatherer loses sync, mark port INACTIVE.
// If PPU controller requests PPU reload, mark port INACTIVE.
if (sync_lost_nxt_MEM || reload_req_MEM) begin
    port_st_nxt_MEM = 'INACTIVE;
end else begin
    // If PPU receives all its cells, port = ACTIVE
    if (wr_oc_eq0_MEM) begin
        port_st_nxt_MEM = 'ACTIVE;
    end
end
end // case: 'HALTED

endcase // case(port_st_MEM)
end // block: port_st_ctrl

always @(ppu_ic_reset_ptr_MEM or
    port_st_MEM or
    wr_oc_eq0_MEM or
    mem_access
    ) begin
    ppu_ic_reset_MEM = 0;

    // Make sure there are no false PPU timeouts.
    if (ppu_ic_reset_ptr_MEM ||
        wr_oc_eq0_MEM ||
```

03/30/00
02:43:07

```
(port_st_MEM != 'ACTIVE') ||  
    mem_access) begin  
    ppu_ic_reset_MEM = 1;  
end  
end // block: port_status_ppu_ic_sched  
  
// DEBUG  
always @(posedge clk) begin  
    if ('INFO_DISPLAY && 'INFO_UNHALTED &&  
        wr_oc_eq0_MEM && (port_st_MEM == 'HALTED)) begin  
        $display ($time, "\tINFO %m:");  
        $display ($time, "\t*** 001: Halted port unhaltd.");  
    end  
end  
  
//  
// rd_word_wr_hd_ptr_eq_wr_hd_ptr[0:3]  
//  
always @(rd_word_FAN_C or  
    wr_hd_ptr_MEM  
    ) begin  
    rd_word_wr_hd_ptr_eq_wr_hd_ptr_MEM =  
        (rd_word_FAN_C['WR_HD_PTR] == wr_hd_ptr_MEM);  
end  
  
//  
// wr_oc_full[0:3]  
//  
always @(posedge clk) begin  
    wr_oc_full_MEM <= (wr_oc_MEM >= wr_oc_full_thresh);  
end  
  
//  
// wr_oc_ov[0:3]  
//  
always @(posedge clk) begin  
    wr_oc_ov_MEM <= (wr_oc_MEM >= wr_oc_ov_thresh);  
end  
  
//  
//  
// Datapath  
//  
//  
//  
//  
// port_st[0:3]  
//  
//
```

ba_cmu_dp

```
always @(posedge clk) begin  
    case (reset_b)  
        0 : port_st_MEM <= 'CLOSED;  
        1 : port_st_MEM <= (port_st_nxt_MEM);  
    endcase // case (reset_b)  
end  
  
// reload_ack[0:3]  
//  
always @(posedge clk) begin  
    case (reset_b)  
        0 : reload_ack_MEM <= 0;  
        1 : reload_ack_MEM <= (reload_ack_nxt_MEM);  
    endcase // case (reset_b)  
end  
  
// ppu_ic[0:3]  
//  
always @(posedge clk) begin  
    case (ppu_ic_reset_MEM)  
        0 : ppu_ic_MEM <= (ppu_ic_MEM + 1);  
        1 : ppu_ic_MEM <= 0;  
    endcase // case (ppu_ic_reset_MEM)  
end  
  
// ppu_ic_ov[0:3]  
//  
always @(posedge clk) begin  
    ppu_ic_ov_MEM <= (ppu_ic_MEM > (4'b0, ppu_timeout, 12'b0));  
end  
  
// rd_port_locked[0:3]  
//  
always @(posedge clk) begin  
    case (reset_b)  
        0 : rd_port_locked_MEM <= 0;  
        1 : rd_port_locked_MEM <= (rd_port_locked_nxt_MEM);  
    endcase // case (reset_b)  
end  
  
// rd_ptrs_reloaded[0:3]  
//  
always @(posedge clk) begin  
    case (reset_b)  
        0 : rd_ptrs_reloaded_MEM <= 0;  
        1 : begin  
            case ((rd_ptrs_reloaded_clear_sel_MEM,  
                rd_ptrs_reloaded_set_sel_MEM))  
                0 : rd_ptrs_reloaded_MEM <= (rd_ptrs_reloaded_MEM);  
            end  
        end  
    end  
end
```

```

always @(posedge clk) begin
    case (reset_b)
        0 : rd_oc_nxt_MEM = 0;
        1 : begin
            case (rd_oc_roll_sel_MEM)
                0 : begin
                    case ({rd_oc_load_sel_MEM, rd_oc_hold_sel_MEM})
                        0 : rd_oc_nxt_MEM = (rd_oc_MEM - 1);
                        1 : rd_oc_nxt_MEM = (rd_oc_MEM);
                        2 : rd_oc_nxt_MEM = (rd_word_FAN_A[RD_TL_PTR]
                            rd_hd_ptr_MEM - 1);
                        3 : rd_oc_nxt_MEM = (rd_word_FAN_A[RD_TL_PTR]
                            rd_hd_ptr_MEM);
                    endcase
                end
                1 : rd_oc_nxt_MEM = (rd_oc_rd_lat);
            endcase // case({rd_oc_roll_sel_MEM,...
        end
        endcase // case({rd_oc_hold_sel_MEM,...
    end
    rd_oc_MEM <= (rd_oc_nxt_MEM & rd_addr_mask);
    rd_oc_eq0_MEM <= ((rd_oc_nxt_MEM & rd_addr_mask) == 0);
end

////////////////////////
// wr_port_locked[0:3]
//
always @(posedge clk) begin
    case (reset_b)
        0 : wr_port_locked_MEM <= 0;
        1 : wr_port_locked_MEM <= (wr_port_locked_nxt_MEM);
    endcase // case(reset_b)
end

////////////////////////
// wr_ptrs_reloaded[0:3]
//
always @(posedge clk) begin
    case (reset_b)
        0 : wr_ptrs_reloaded_MEM <= 0;
        1 : begin
            case ({wr_ptrs_reloaded_clear_sel_MEM,
                wr_ptrs_reloaded_sel_MEM})
                0 : wr_ptrs_reloaded_MEM <= (wr_ptrs_reloaded_MEM);
                1 : wr_ptrs_reloaded_MEM <= 1;
                2, 3 : wr_ptrs_reloaded_MEM <= 0;
            endcase // case({wr_ptrs_reloaded_clear_sel_MEM,...
        end
        endcase // case({reset_b, wr_ptrs_reloaded_clear_sel, ...
    end // always @ (posedge clk)

////////////////////////
// wr_oc[0:3], wr_oc_eq0[0:3]
//
always @(posedge clk) begin

```

03/30/00
02:43:07

ba_cmu_port_dp.v

7

```
0 : wr_oc_nxt_mem = 0;
1 : begin
  case ((wr_oc_load_sel_mem, wr_oc_hold_sel_mem))
    0 : wr_oc_nxt_mem = (wr_oc_mem + 1);
    1 : wr_oc_nxt_mem = (wr_oc_mem);
    2 : wr_oc_nxt_mem = (wr_tl_ptr_mem -
      rd_word_fan_b['wr_hd_ptr'] + 1);
    3 : wr_oc_nxt_mem = (wr_tl_ptr_mem -
      rd_word_fan_b['wr_hd_ptr']);
  endcase // case((wr_oc_load_sel_mem, wr_oc_hold_sel_mem))
end
endcase // case((wr_oc_hold_sel_mem, wr_oc_load_sel_mem)
wr_oc_mem <= (wr_oc_nxt_mem & wr_addr_mask);
wr_oc_eq0_mem <= ((wr_oc_nxt_mem & wr_addr_mask) == 0);
end

////////////////////
// wr_hd_ptr[0:3]
//
always @(posedge clk) begin
  case (wr_hd_ptr_load_sel_mem) // {load}
    0 : wr_hd_ptr_mem <= (wr_hd_ptr_mem);
    1 : wr_hd_ptr_mem <= (rd_word_fan_c['wr_hd_ptr']);
  endcase
end

////////////////////
// wr_tl_ptr[0:3], wr_tl_ptr_shadow[0:3]
//
always @(wr_tl_ptr_load_sel_mem or
  wr_tl_ptr_hold_sel_mem or
  wr_tl_ptr_mem or
  wr_addr_mask or
  rd_word_fan_c
) begin
  case ((wr_tl_ptr_load_sel_mem, wr_tl_ptr_hold_sel_mem))
    0 : wr_tl_ptr_mem_nxt = ((wr_tl_ptr_mem & wr_addr_mask) |
      ((wr_tl_ptr_mem + 1) & wr_addr_mask));
    1 : wr_tl_ptr_mem_nxt = (wr_tl_ptr_mem);
    2, 3 : wr_tl_ptr_mem_nxt = (rd_word_fan_c['wr_tl_ptr']);
  endcase
end

always @(posedge clk) begin
  wr_tl_ptr_mem <= wr_tl_ptr_mem_nxt;
end

always @(posedge clk) begin
  case ((wr_tl_ptr_shadow_load_sel_mem, wr_tl_ptr_shadow_hold_sel_mem))
    0 : wr_tl_ptr_shadow_mem <= (wr_tl_ptr_mem_nxt);
    1 : wr_tl_ptr_shadow_mem <= (wr_tl_ptr_shadow_mem);
    2, 3 : wr_tl_ptr_shadow_mem <= (rd_word_fan_c['wr_tl_ptr']);
  endcase
end

endmodule // ba_cmu_core
```

```
module ba_cmu_var_delay
(clk, reset_b, in, delay, out);

parameter WIDTH = 1;
parameter DELAY_TYPE = 0;

input clk;
input reset_b;
input [WIDTH-1:0] in;
input [2:0] delay;
output [WIDTH-1:0] out;

reg [WIDTH-1:0] out;
reg [WIDTH-1:0] in_P1;
reg [WIDTH-1:0] in_P2;
reg [WIDTH-1:0] in_P3;
reg [WIDTH-1:0] in_P4;
reg [WIDTH-1:0] in_P5;
reg [WIDTH-1:0] in_P6;
reg [WIDTH-1:0] in_P7;

always @(posedge clk) begin
  if (!reset_b) begin
    in_P1 <= 0;
    in_P2 <= 0;
    in_P3 <= 0;
    in_P4 <= 0;
    in_P5 <= 0;
    in_P6 <= 0;
    in_P7 <= 0;
  end else begin
    in_P1 <= in;
    in_P2 <= in_P1;
    in_P3 <= in_P2;
    in_P4 <= in_P3;
    in_P5 <= in_P4;
    in_P6 <= in_P5;
    in_P7 <= in_P6;
  end
end

always @(delay or
  in or
  in_P1 or
  in_P2 or
  in_P3 or
  in_P4 or
  in_P5 or
  in_P6 or
  in_P7
) begin
  case (DELAY_TYPE)
    'NORM_DELAY: begin
      case (delay)
        0 : out = in;
        1 : out = in_P1;
        2 : out = in_P2;
        3 : out = in_P3;
        4 : out = in_P4;
      endcase
    end
  endcase
end
```

03/30/00
02:43:07

ba_cmu_port_dp.v

8

```
5 : out = in_P5;
6 : out = in_P6;
7 : out = in_P7;
    endcase // case(delay)
end

'OR_PREV_DELAY: begin
    case (delay)
        0 : out = 0;
        1 : out = in;
        2 : out = (in | in_P1 | in_P2);
        3 : out = (in | in_P1 | in_P2 | in_P3);
        4 : out = (in | in_P1 | in_P2 | in_P3 | in_P4);
        5 : out = (in | in_P1 | in_P2 | in_P3 | in_P4 | in_P5);
        6 : out = (in | in_P1 | in_P2 | in_P3 | in_P4 | in_P5 | in_P6);
        7 : out = (in | in_P1 | in_P2 | in_P3 | in_P4 | in_P5 | in_P6);
    endcase // case(delay)
end

default: out = 0;

    endcase // case(DELAY_TYPE)

end // block: delay_comb

endmodule // ba_cmu_var_delay

=====
// This Software and Related Documentation contain the trade secret
// and confidential information of Celox Communications, Inc.
// Copyright (c) 1999 Celox Communications, Inc., St. Louis, MO
// Unpublished.
// All Rights Reserved.
//
// Restricted Rights Legend: The Software and Documentation
// are provided with Restricted Rights. Use, Duplication, or Disclosure
// by the Government is Subject to Restrictions as Set Forth in
// Paragraph (c)(1)(ii) of the Rights in Technical Data and
// Computer Software Clause at DFARS 252.227-7013. Contractor/Manufacturer
// is Celox Communications, Inc., 4041 Forest Park Blvd., St. Louis, MO 63108.
```


03/28/00
10:52:02

define ba_cmu_vh_cj

```
'define NUM_PORTS 4

// INACTIVE means that the port needs to finish up atomic cell/packet transactions
'define INACTIVE 0
// CLOSING means that the port needs to write out pointers once more
'define CLOSING 1
// CLOSED means that no activity should be done on the port at all
'define CLOSED 2
// RELOADING means that the port should only attempt to reload all 4 pointers
'define RELOADING 3
// ALIVE means the port is normal, read/writing pointers/data
'define ACTIVE 4
// HALTED means that data will not be written to the port, all else normal
'define HALTED 5

'define NORM_DELAY 0
'define OR_PREV_DELAY 1

'define CTRL_CELL_LEN 61:48
'define CTRL_SIGNATURE 35:28
'define CTRL_CI_BIT 40
'define CTRL_PPU 39:38
'define CTRL_SOP 46
'define CTRL_PIDS 41
'define SCHED_SIGNATURE 63:56
'define SCHED_CELL_LEN 12:0
'define SCHED_B_BIT 29

'define RD_HD_PTR 51:32
'define RD_TL_PTR 19:0
'define WR_HD_PTR 51:32
'define WR_TL_PTR 19:0

'define RD_HD_WR_TL_C1_PPU0_ADDR 0 // written pointers
'define RD_HD_WR_TL_C2_PPU0_ADDR 1 // written pointers
'define WR_HD_RD_TL_C1_PPU0_ADDR 16 // read pointers
'define WR_HD_RD_TL_C2_PPU0_ADDR 17 // read pointers
'define RD_HD_WR_TL_C1_PPU1_ADDR 32 // written pointers
'define RD_HD_WR_TL_C2_PPU1_ADDR 33 // written pointers
'define WR_HD_RD_TL_C1_PPU1_ADDR 48 // read pointers
'define WR_HD_RD_TL_C2_PPU1_ADDR 49 // read pointers
'define SCRATCH_ADDR 64

'define FIFO_SIZE 256
'define FIFO_ADDR 8
'define FIFO_CALC 'FIFO_ADDR+2

'define no_msg 1 // turns off IBM library warnings

'define INFO_DISPLAY 1
'define INFO_REDIR 0
'define INFO_UNHALTED 1
```